TSN - 150/PCI

Modulo ETN Master con interfaccia PCI

MANUALE UTENTE

MANTSN-150PCI - Rev. 3 - 07.GEN.2010

Stampato in Italia / Printed in Italy

<table>
<tr><th colspan="6">Archivio</th></tr>
<tr><td colspan="2">Codice Documento</td><td colspan="2">Revisione</td><td colspan="2">Data</td></tr>
<tr><td colspan="2">MANTSN-150PCI</td><td colspan="2">3</td><td colspan="2">07.01.2010</td></tr>
<tr><th colspan="6">Scheda</th></tr>
<tr><td colspan="3">TSN-150/PCI</td><td colspan="3">PCBM0050.150A</td></tr>
<tr><td colspan="3">Nome</td><td colspan="3">Revisione Circuito Stampato</td></tr>
<tr><th colspan="6">Firme</th></tr>
<tr><td colspan="2">Gti</td><td colspan="2">Rro</td><td colspan="2">Pfr</td></tr>
<tr><td colspan="2">Redatto</td><td colspan="2">Verificato</td><td colspan="2">Approvato</td></tr>
<tr><td colspan="6">DOCUMENTO CON FIRMA ELETTRONICA - DOCUMENT WITH ELECTRONIC SIGNATURE</td></tr>
</table>

Headquarters:
Via della Tecnica, 16/18
23875 Osnago (LC) - ITALY
PHONE +39 039 5969100 - FAX +39 039 5969124
E-mail : info@tecnint.it
: support@tecnint.it

INDICE

INDICE DELLE FIGURE

INDICE DELLE TABELLE

1. VERIFICHE PRELIMINARI

Operazioni da effettuare prima dell'utilizzo del modulo:

- Verificare che l'imballaggio del modulo non sia danneggiato. In caso contrario, aprire il contenitore ed ispezionare il contenuto in presenza del corriere.
- Conservare l'imballaggio per eventuali trasporti futuri.
- Sempre subito dopo l'apertura ispezionare visivamente il modulo e controllare che il circuito stampato, i connettori frontali e tutti i componenti siano correttamente installati e non danneggiati durante il trasporto.
- Leggere attentamente il presente manuale.

Solo quando tutti i punti precedenti siano soddisfatti è possibile procedere all'utilizzo del modulo.

2. INTRODUZIONE

Il modulo TSN-150/PCI è un nuovo modulo master ETN ad alte prestazioni per PC IBM compatibili con interfaccia PCI.

L'interfaccia PCI è realizzata utilizzando il componente PLX Technology PCI 9052. Si rimanda al manuale di tale componente per le relative informazioni.

Il master ETN presente sul modulo è adattato specificamente per l'uso in applicazioni basate su PC. Mediante l'interfaccia ETN, trasmette e riceve dati dai vari moduli SLAVE della famiglia TSR collegati in rete. Il collegamento con i moduli di tipo SLAVE può essere effettuato con cavo eletrico o con fibra ottica plastica, vetro o siliconica (HCS). Il sistema ETN è un sistema distribuito progettato specificamente per l'automazione industriale ed i sistemi di controllo ed acquisizione dati. Per ulteriori informazioni riguardo al sistema ETN si rimanda al capitolo 7 ed alle seguenti pubblicazioni tecnint HTE :

- "Il sistema ETN"
- "Introduzione al sistema ETN"(app. note #2)
- "Calcolo della banda passante di un sistema ETN"(app. note #3)
- "La topologia del sistema ETN"(app. note #4)

3. INFORMAZIONI D'ORDINE

Per eventuali ordini del modulo in oggetto, fare riferimento al codice indicato nella seguente tabella:

VERSIONE	**Codice d'ordine**	**Disponibilità**
Versione standard (ETN max. 6 Mbit/s)	TSN-150/PCI	Si
Versione con ETN a 12 Mbit/s max.	TSN-150/PCI12	Si
Versione Std con fibra ottica vetro	TSN-150/PCIOV	Su richiesta
Versione Std con fibra ottica plastica	TSN-150/PCIOP	Su richiesta
Versione Std con fibra ottica HCS (hard clad silica)	TSN-150/PCIHCS	Su richiesta
Versione 12M con fibra ottica vetro	TSN-150/PCI12OV	Su richiesta
Versione 12M con fibra ottica plastica	TSN-150/PCI12OP	Su richiesta
Versione 12M con fibra ottica HCS (hard clad silica)	TSN-150/PCI12HCS	Su richiesta

Tabella 3-1: Informazioni d'ordine

4. DESCRIZIONE TECNICA

4.1. CARATTERISTICHE TECNICHE

Le specifiche del modulo TSN-150/PCI sono riportate nella seguente tabella.

Interfaccia ETN MASTER	Baud rate selezionabile da 93Khz fino a 6Mbit/s (opzionale fino a 12Mbit/s) . 2 linee RS485 optoisolate a 750Volt. 1 linea di trasmissione in fibra ottica plastica, vetro o siliconica (opzionale). 1 linea di ricezione in fibra ottica plastica, vetro o siliconica (opzionale).
Interfaccia PCI	Di tipo slave realizzata utilizzando il componente PLX Technology PCI 9052 (compliant con specifiche PCI v2.1) Bus dati locale 8 bit. Decodifica del modulo nello spazio di memoria e nello spazio di IO Modulo adatto al bus PCI a 5V
Memoria ETN	RAM dual ported 32kx8 condivisa da PC (via BUS PCI) e MASTER ETN, logicamente divisa in due buffer di 16Kbyte ciascuno.
Memoria Eeprom	Di tipo seriale da 1Mbit (64x16 Bits)
Interrupt	Generazione programmabile di un interrupt su BUS PCI alla fine della trasmissione dei record ETN presenti nel buffer.
Reset	Tramite linea di reset del BUS PCI.
Connettori	9 poli maschio per il collegamento seriale in RS485 per il BUS ETN. 1 connettore per trasmissione mediante fibra ottica (opzionale). 1 connettore per ricezione mediante fibra ottica (opzionale). 1 connettore EDGE 62x2 pin per montaggio della scheda su bus PCI.
Led	2 LED utenti disponibili sul frontale
Alimentazione	5V ± 5% @ 600mA.
Temperatura di lavoro	da 0°C a 60°C.
Temperatura di deposito	da -25°C a 85°C.
Umidità	dal 5 al 90 % senza condensa.
Dimensioni	125 x 90 mm circa.

Tabella 4-1: Specifiche del modulo TSN-150/PCI

4.2. ANALISI A BLOCCHI DEL MODULO

4.2.1. Introduzione

In questo capitolo viene riportata una sommaria descrizione a blocchi del modulo.

4.2.2. Schema a blocchi

La figura seguente riporta lo schema a blocchi semplificato del modulo TSN-150/PCI.

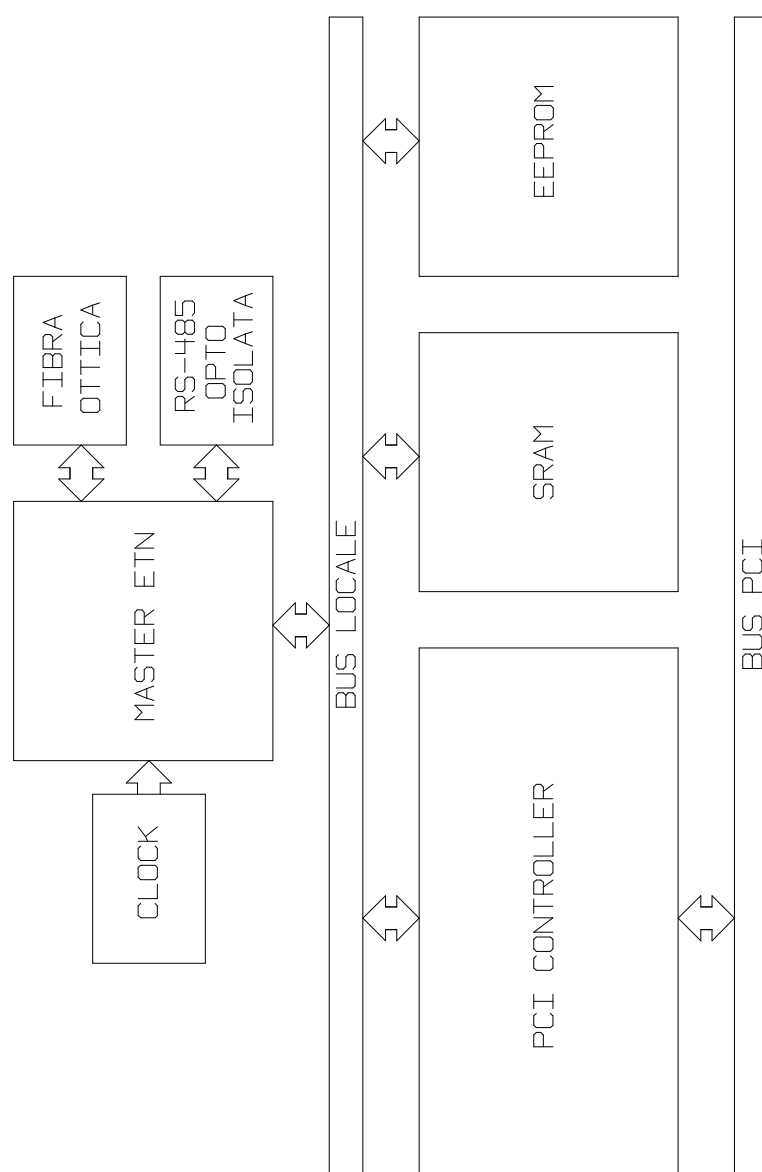



Figura 4.1: Schema a blocchi

4.2.3. Interfaccia PCI

L'interfaccia PCI è di tipo slave (target) ed è realizzata utilizzando il componente PLX Technology PCI 9052 (compliant con specifiche PCI v2.1). Il Bus dati locale è a 8 Bit. Per ulteriori informazioni si rimanda al manuale del componente citato.

4.2.4. Interfaccia ETN Master

Il modulo TSN-150/PCI è un modulo ETN Master. L'interfaccia ETN presente a bordo ha le seguenti caratteristiche principali (per maggiori informazioni si rimanda al Cap. 7 e ai documenti citati nel Cap. 2):

- implementazione in fibra ottica plastica, vetro o siliconica oppure tramite driver RS-485 (2 canali fisici).
- ha alimentazione e segnali galvanicamente isolati rispetto al resto del modulo; infatti il modulo TSN-150/PCI è alimentato con una tensione pari a +5 V prelevata dal PC nel quale è inserito, ma è provvisto in un DC/DC converter che fornisce un'alimentazione isolata per i buffer della linea seriale ETN.
- segnali disponibili sul connettore J4 per le 2 linee seriali RS-485 o sui connettori OPT2 (ricezione) e OPT1 (trasmissione) per la linea in fibra ottica.
- velocità di trasmissione configurabile tramite registro di configurazione fino a 12 Mbit/s.

4.2.5. Memorie Ram statica dual-ported e memoria Eeprom

La memoria RAM presente al bordo del modulo è di tipo statico, da 32kx8 ed è condivisa da PC (via BUS PCI) e MASTER ETN, logicamente divisa in due buffer di 16 Kbyte ciascuno.
La memoria Eeprom è invece di tipo seriale, da 1Mbit, organizzata secondo un array da 64x16 Bits.

4.3. LAYOUT DEL MODULO

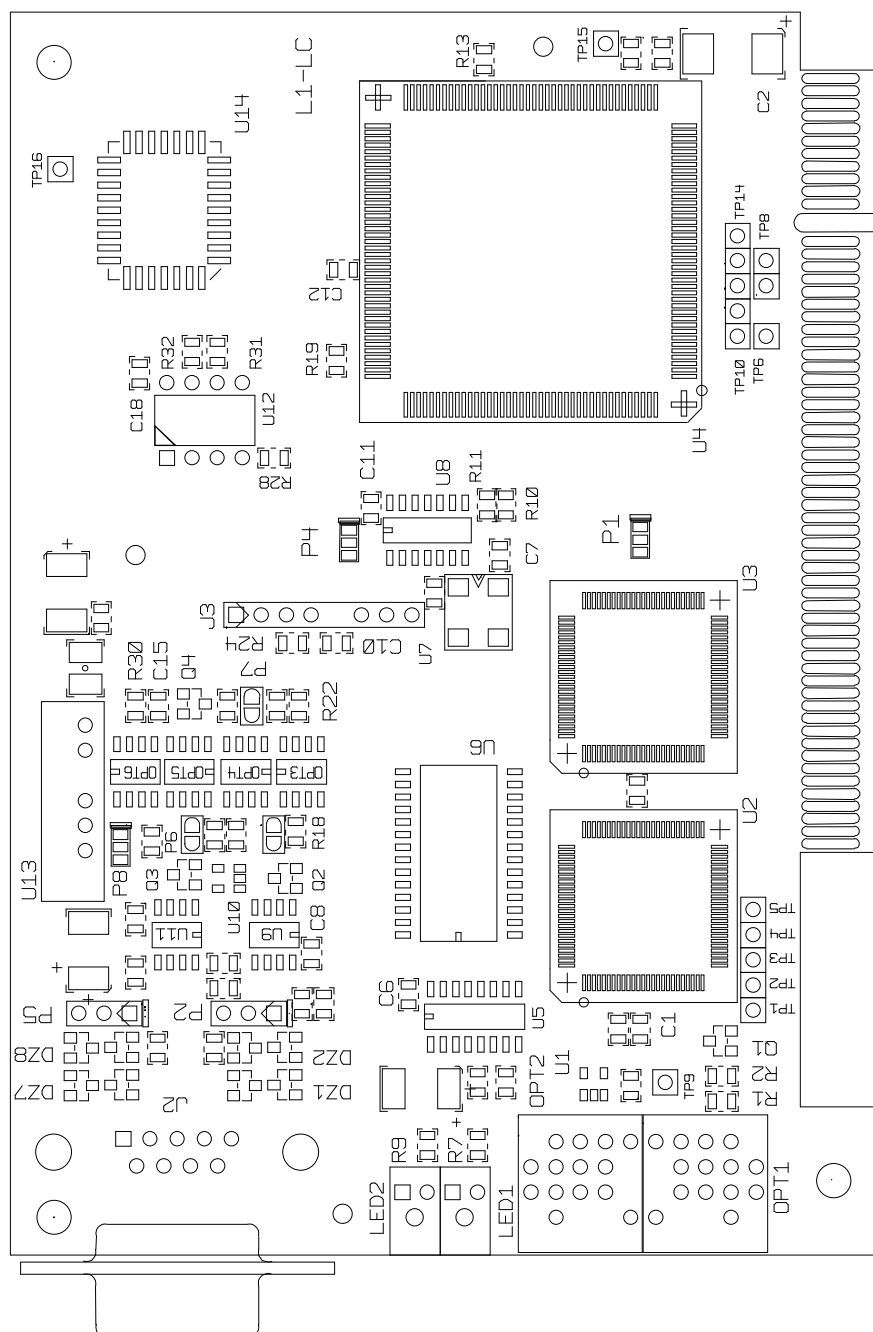



Figura 4.1: Layout modulo lato componenti

5. PONTICELLI, CONNETTORI E LED

5.1. PONTICELLI

5.1.1. Introduzione

La seguente tabella riassume il significato dei ponticelli presenti sul modulo TSN-150/PCI. Per la posizione sul modulo degli unici due ponticelli P2 e P5 configurabili dall'utente, si faccia riferimento alla figura 4-1.

PONTICELLO	FUNZIONE
Ponticelli P2 e P5	Permettono di inserire le resistenze di terminazione sulle linee RS485 del bus seriale ETN (per entrambi 1-2 resistenza inserita, 2-3 resistenza disinserita). Il ponticello P5 risulta associato alla linea ETN 0 (Pin 1 e 2 del connettore a 9 poli) mentre il ponticello P2 alla linea ETN 1 (pin 4 e 5 del connettore a 9 poli).
Ponticelli P1, P3, P4, P6 e P7	Riservati al costruttore. **Questi ponticelli non devono essere modificati dall'utente.**

Tabella 5-1: Ponticelli del modulo TSN-150/PCI

5.1.2. Ponticello P2: resistenza di terminazione linea ETN RS-485 1

Con il ponticello P2 è possibile inserire la resistenza di terminazione sulla linea RS-485 numero 1 del bus seriale ETN, secondo quanto riportato nella seguente tabella.

P2	Resistenza di terminazione linea 1
1—2 3	Inserita
1 2—3	Disinserita

Tabella 5-2: Ponticello P2 - resistenza di terminazione linea ETN 1

5.1.3. Ponticello P5: resistenza di terminazione linea ETN RS-485 0

Con il ponticello P5 è possibile inserire la resistenza di terminazione sulla linea RS485 numero 0 del bus seriale ETN, secondo quanto riportato nella seguente tabella.

P5	Resistenza di terminazione linea 0
1—2 3	Inserita
1 2—3	Disinserita

Tabella 5-3: Ponticello P5 - resistenza di terminazione linea ETN 0

5.1.4. Ponticello P1, P3, P4, P6 e P7

Ponticelli di uso riservato al costruttore. Questi ponticelli non devono essere modificati dall'utente.

5.2. CONNETTORI

5.2.1. Introduzione

La seguente tabella illustra il significato dei connettori presenti sul modulo. Per la posizione sul modulo dei connettori si faccia riferimento alla figura 4-1.

CONNETTORE	FUNZIONE
Connettori J3	Connettore riservato al costruttore
Connettori J1 e J4	Connettori PCI
Connettore J2	Connettore del bus seriale ETN
OPT1	Connettore di trasmissione per fibra ottica plastica, vetro o siliconica
OPT2	Connettore di ricezione per fibra ottica plastica, vetro o siliconica

Tabella 5-4: Connettori del modulo TSN-150/PCI

5.2.2. Connettore J2: ETN RS-485

La seguente tabella descrive la corrispondenza tra i pin ed i segnali del connettore RS-485 J2 dell'interfaccia ETN.

PIN	SEGNALE
1	S+ (linea 0)
2	S- (linea 0)
3	N.C.
4	S+ (linea 1)
5	S- (linea 1)
6	VSS
7	VSS
8	VSS
9	VSS

Tabella 5-5: Connettore J2 bus ETN (RS-485)

5.3. LED

A bordo del modulo sono presenti 2 Led il cui utilizzo è configurabile dall'utente. Per la loro posizione si faccia riferimento alla figura 4-1. Tali Led vengono pilotati direttamente dal controllore PCI. Le seguenti tabelle illustrano come pilotare tali Led, per maggiori informazioni invece sul controllore PCI PLX Technology PCI 9052, si rimanda al manuale di tale componente.

LED	Pin controllore PCI associato	LED ON se livello logico Pin
LED1	139	ALTO (1)
LED2	138	ALTO (1)

Tabella 5-6: Led

6. RISORSE INTERNE

Il PC può accedere alle seguenti risorse del modulo TSN-150/PCI.

- La memoria si scambio ETN.
- I registri interni.

6.1. Caratteristiche di funzionamento del modulo

Il modulo master ETN si interfaccia con l' utente attraverso una memoria e alcuni registri di I/O.
La memoria di scambio consente all' utente di inviare e ricevere dati sulla rete ETN, mentre i registri permettono di controllare alcune funzioni del master ETN.

6.2. La memoria di scambio

La memoria di scambio dati con il bus ETN ha una dimensione di **32K** ed è suddivisa in due buffer di **16kbyte** ciascuno, in ognuno dei quali è possibile allocare da **0** a **1022** record di scambio.
Un buffer è detto **attivo**, ed è utilizzato dal master ETN per trasmettere e ricevere dati sulla rete ETN, mentre l' altro buffer è detto **passivo** ed è accessibile dal lato PC.
La procedura di colloquio con gli slave del bus ETN prevede che l' applicativo PC prepari il buffer **passivo** con i dati necessari ed attivi quindi la comunicazione scambiando i ruoli tra i buffer: l' attivo diventa passivo, e disponibile al PC, mentre il passivo diventa attivo e disponibile alla rete ETN. Il PC può così leggere i dati disponibili inviati dagli slave mentre il master ETN può aggiornare gli slave con i nuovi valori stabiliti dal PC.
La commutazione tra i due buffer avviene in corrispondenza di una scrittura sul bit di selezione buffer. Va notato che gli indirizzi di memoria non cambiano: il PC ed il master ETN accedono sempre agli stessi indirizzi relativi, anche se fisicamente la memoria utilizzata è diversa.
Il buffer disponibile all' utente va dall' indirizzo relativo **0x0000** all' indirizzo **0x3FFF**, mentre il master ETN vede il proprio buffer a partire dall' indirizzo relativo **0x4000** all' indirizzo **0x7FFF**.
Nell' area di scambio è previsto un record riservato, dall' indirizzo relativo **0x0000** all' indirizzo **0x000F**, destinato ad usi interni e perciò non utilizzabile per le transazioni normali: tutti i byte di questo record devono essere inizializzati a 0, in entrambi i buffer di scambio.
Il buffer con i dati disponibili all' utente si trova a partire dall'indirizzo relativo **0x0010** (primo byte del primo record) fino all'indirizzo **0x3FEF** (ultimo byte dell'ultimo record).
Mentre il buffer usato dal Master ETN per il trasferimento dei dati si trova a partire dall'indirizzo relativo **0x4010** (primo byte del primo record) fino all'indirizzo **0x7FEF** (ultimo byte dell'ultimo record).

I record, ognuno dei quali occupa **16** byte, devono essere inizializzati in ogni buffer in modo consecutivo; per indicare la fine dei record presenti nel buffer è necessario inizializzare tutto il record successivo con il valore **0x00**.

Ogni record contiene tutte le informazioni necessarie per il colloquio con un modulo SLAVE, è possibile indirizzare lo stesso modulo con più record per aumentarne la velocità di rinfresco rispetto agli altri o per scambiare dati con più risorse o canali all'interno dello stesso modulo. Gli errori rilevati sul bus (memorizzati nel flag di errore) possono essere causati da una delle seguenti anomalie:

1. errato indirizzo del modulo MASTER o SLAVE
2. errato TIPO di modulo SLAVE
3. errore di parità in trasmissione o ricezione
4. errore di frame in trasmissione o ricezione
5. errore di OVERRUN in trasmissione o ricezione
6. errore di LRC in trasmissione o ricezione

6.2.1. Accesso alla memoria di scambio

Da PC è possibile accedere ai buffer **passivo ed attivo** della memoria ETN sulla scheda TSN150/PCI tramite due spazi di indirizzamento differenti, con parallelismo a Byte:

1. Attraverso lo spazio di indirizzamento di memoria del controller PCI e quindi effettuando un normale ciclo di scrittura/lettura all'indirizzo selezionato

2. In modalità emulazione di TSN150/ISA, attraverso lo spazio di I/O del controller PCI.

In modalità si emulazione TSN150/ISA si accede alla memoria utilizzando i primi 3 registri interni all'offset **0x0000**, **0x0001** e **0x0002**.

Si tratta di scrivere l'indirizzo della memoria (15 bit) al quale si desidera accedere spezzato in 2 byte nei due registri ADDRESS LSB (offset 0) e ADDRESS MSB (offset 1) e poi scrivere o leggere il byte corrispondente accedendo rispettivamente in scrittura o in lettura al registro DATA (offset 2).
L'hardware provvede ad incrementare l'indirizzo memorizzato dopo ogni accesso al registro DATA, quindi un successivo accesso al registro DATA viene effettuato all'indirizzo dell'accesso precedente + 1.

Si noti che il registro LSB è a 8 bit mentre il MSB è a 7 bit, il bit più significativo (Bit7)è don't care.
I registri ADDRESS LSB e MSB sono accessibili anche in lettura.

6.3. REGISTRI

I registri del modulo TSN-150/PCI sono accessibili da PC nello spazio di I/O a partire dall'indirizzo base impostato. La seguente tabella riporta l'elenco completo dei registri con l'offset rispetto all'indirizzo base di I/O.

OFFSET	REGISTRO
0	Registro ADDRESS LSB memoria ETN
1	Registro ADDRESS MSB memoria ETN
2	Registro DATA memoria ETN
3	Registro di configurazione standard
4	registro reset interrupt/start ETN
5	registro di configurazione esteso

Tabella 6-1: Mappa di memoria dei registri interni nello spazio di I/O del PC

Tutti i registri del modulo TSN-150/PCI sono a 8 bit, quindi tutte le operazioni di lettura e scrittura devono essere effettuate a byte.

6.3.1. Registro ADDRESS LSB memoria ETN

Il registro ADDRESS LSB memoria ETN è un registro a 8 bit, accessibile sia in scrittura che in lettura. Rappresenta la parte meno significativa dell'indirizzo di accesso alla memoria ETN. Il significato dei singoli bit in scrittura e lettura è riportato nella seguente tabella.

BIT	ACCESSO	DESCRIZIONE
0	R/W	bit di indirizzo 0 memoria ETN
1	R/W	bit di indirizzo 1 memoria ETN
2	R/W	bit di indirizzo 2 memoria ETN
3	R/W	bit di indirizzo 3 memoria ETN
4	R/W	bit di indirizzo 4 memoria ETN
5	R/W	bit di indirizzo 5 memoria ETN
6	R/W	bit di indirizzo 6 memoria ETN
7	R/W	bit di indirizzo 7 memoria ETN

Tabella 6-2: Struttura del registro ADDRESS LSB memoria ETN

All’accensione e dopo il reset del PC i bit del registro ADDRESS LSB memoria ETN sono uguali a 0. Il registro si incrementa di uno ad ogni accesso al registro DATA memoria ETN da parte del PC.

6.3.2. Registro ADDRESS MSB memoria ETN

Il registro ADDRESS MSB memoria ETN è un registro a 8 bit, accessibile sia in scrittura che in lettura. Rappresenta la parte più significativa dell’indirizzo di accesso alla memoria ETN. Il significato dei singoli bit in scrittura e lettura è riportato nella seguente tabella.

BIT	ACCESSO	DESCRIZIONE
0	R/W	Bit di indirizzo 8 memoria ETN
1	R/W	Bit di indirizzo 9 memoria ETN
2	R/W	Bit di indirizzo 10 memoria ETN
3	R/W	Bit di indirizzo 11 memoria ETN
4	R/W	Bit di indirizzo 12 memoria ETN
5	R/W	Bit di indirizzo 13 memoria ETN
6	R/W	Bit di indirizzo 14 memoria ETN
7	-	Non utilizzato

Tabella 6-3: Struttura del registro ADDRESS MSB memoria ETN

All’accensione e dopo il reset del PC i bit del registro ADDRESS MSB memoria ETN sono uguali a 0. Il registro si incrementa di uno ad ogni riporto proveniente dal registro ADDRESS LSB.

6.3.3. Registro DATA memoria ETN

Il registro DATA memoria ETN è un registro a 8 bit, accessibile sia in scrittura che in lettura che rappresenta la locazione di memoria ETN attualmente puntata dall’indirizzo presente nei registri ADDRESS MSB + LSB. L’accesso in lettura o scrittura a questo registro produce un incremento unitario dell’indirizzo puntato dai sopra detti registri, che viene effettuato successivamente all'accesso.

6.3.4. Registro di configurazione standard

Il registro di configurazione è un registro a 8 bit, accessibile sia in scrittura che in lettura. Il significato dei singoli bit in scrittura e lettura è riportato nella seguente tabella.

BIT	ACCESSO	DESCRIZIONE
0	R/W	= 1 abilita il Bus ETN
1	R/W	= 1 abilita la generazione dell'interrupt su bus PCI
2	R/W	set buffer memoria ETN
3	-	Riservato
4	-	Riservato
5	-	Riservato
6	-	Riservato
7	R	= 1 a fine trasmissione dei record presenti all’interno del buffer ETN

Tabella 6-4: Struttura del registro di configurazione standard

All’accensione e dopo il Reset del PC i bit del registro di configurazione standard sono uguali a 0.

6.3.4.1. Bit 0 abilitazione ETN

Se impostato a 1 abilita il Bus ETN, in tal caso è possibile trasmettere record presenti nel buffer di memoria attivo accedendo al registro reset interrupt/start ETN; il reset del bit provoca l’interruzione della trasmissione in corso.

6.3.4.2. Bit 1 abilitazione interrupt di fine trasmissione

Se impostato a 1 abilita la generazione di una richiesta di interrupt su bus PCI. La disabilitazione della richiesta pendente deve essere fatta dal PC con un accesso al registro reset interrupt/start ETN. Si noti che tale accesso nel caso in cui il bit di abilitazione ETN (bit 0 del registro di configurazione standard) sia attivo, oltre che resettare l’interrupt provoca un nuovo start della trasmissione sul bus.

6.3.4.3. Bit 2 selezione buffer della memoria ETN in trasmissione

Il bit 2 seleziona il buffer della memoria ETN che andrà in trasmissione. Se impostato a 0 i record trasmessi sono quelli presenti nel buffer 0, se impostato a 1 i record trasmessi sono quelli presenti nel buffer 1. Mentre un buffer è in trasmissione l’altro è disponibile alla CPU del PC.

6.3.4.4. Bit 3, 4, 5, 6

Questi Bit sono riservati e non devono essere utilizzati.

6.3.4.5. Bit 7 status di fine transazioni ETN

Questo bit a sola lettura viene posto al valore 1 dopo la trasmissione dell'ultimo record presente nel buffer di memoria attivo.

6.3.5. Registro Reset Interrupt/Start ETN

Il registro Reset Interrupt/Start ETN è un registro a 8 bit (il dato non è considerato), accessibile sia in lettura che in scrittura. Un accesso a tale registro permette di resettare una eventuale richiesta di interrupt pendente, inoltre se il bus ETN è abilitato, tale accesso provoca lo start della trasmissione sul bus.

6.3.6. Registro di Configurazione Esteso

Il registro di configurazione esteso è un registro a 8 bit, accessibile sia in scrittura che in lettura. Il significato dei singoli bit in scrittura e lettura è riportato nella seguente tabella.

BIT	ACCESSO	DESCRIZIONE
0	R/W	Selezione frequenza di trasmissione
1	R/W	Selezione frequenza di trasmissione
2	R/W	Selezione frequenza di trasmissione
3	R/W	Riservato
4	R/W	Riservato
5	R/W	= 1 selezione modalità di rinfresco bus ETN
6	R/W	= 1 selezione modalità di rinfresco bus ETN
7	-	Riservato

Tabella 6-5: Struttura del registro di configurazione esteso

All'accensione e al reset del PC i bit del registro di configurazione esteso sono uguali a 0.

6.3.6.1. Bit 0, 1 e 2 selezione frequenza di trasmissione

L’impostazione della frequenza di trasmissione si esegue dai bit 0-2 del registro di configurazione esteso.
La frequenza di 12 MHz è attiva per la versione del modulo a 12 Mbit/s, altrimenti è da intendere uguale a 6 Mbit/s.
La selezione della frequenza di trasmissione è ottenuta impostando i bit 0, 1 e 2 come indicato dalla tabella seguente.

Selezione frequenza di trasmissione			
Bit 2	bit 1	Bit 0	Velocità
0	0	0	12 Mbit/s (6 Mbit/s)
0	0	1	6 Mbit/s
0	1	0	3 Mbit/s
0	1	1	1,5 Mbit/s
1	0	0	750 Kbit/s
1	0	1	375 Kbit/s
1	1	0	187 Kbit/s
1	1	1	93,75 Kbit/s

Tabella 6-6: Selezione della velocità di trasmissione sul bus ETN

L'aggiornamento dello stato delle uscite e l'acquisizione dello stato degli ingressi dei vari moduli SLAVE avviene ad una velocità che dipende dalla frequenza selezionata sui vari moduli della rete ossia dalla frequenza di lavoro della rete ETN.

La relativa casistica è descritta nella seguente tabella.

TRANSAZIONI/secondo	FREQUENZA DEL BUS
80000	12 Mbit/s
40000	6 Mbit/s
20000	3 Mbit/s
10000	1,5 Mbit/s
5000	750 Kbit/s
2500	375 Kbit/s
1250	187,5 Kbit/s
625	93,75 Kbit/s

Tabella 6-7: Transazioni/secondo tra il modulo TSN-150/PCI ed i vari moduli SLAVE

6.3.6.2. Bit 5 e 6 selezione modalità di rinfresco rete ETN

Questi bit selezionano la modalità di rinfresco della rete ETN. Sono possibili tre situazioni di rinfresco come indicato nella Tabella 6-8.

Selezione modalità di rinfresco		
bit 6	**bit 5**	**rinfresco**
0	0	rinfresco tramite accesso al registro reset interrupt/start ETN
0	1	rinfresco automatico per la durata di un secondo
1	x	rinfresco continuo

Tabella 6-8: Selezione modalità rinfresco automatico

I moduli slave ETN (TSR-31, TSR-40, ecc.) sono dotati di un dispositivo watchdog che, nel caso non ricevano trasmissioni per più di 100 ms., ne blocca il funzionamento e disattiva le uscite finché non viene ricevuta una nuova trasmissione. Se il programma software che gestisce il bus ETN imposta la modalità di rinfresco tramite accesso al registro reset interrupt/start ETN, e poi non effettua il rinfresco nel tempo limite, provoca l'attivazione dei watchdog dei moduli slave.
Il watchdog e' un importante fattore di sicurezza nelle applicazioni real time, ma in altri tipi di applicazioni, ovvero in applicazioni in cui il tempo di acquisizione dei dati non e' critico ed è superiore al tempo di intervento del watchdog, esso può costituire un inconveniente. Per superare il problema sono disponibili le modalità di rinfresco automatico: una volta avviata la trasmissione con il primo accesso al registro reset interrupt/start ETN, il ciclo di trasmissione/ricezione non termina alla fine dei record nel buffer, ma riprende immediatamente dal primo record. In questo caso, non viene generato l'interrupt di fine trasmissione e il bit 7 del registro di configurazione standard rimane a 0.
Per riprendere il controllo del bus ETN, il programma deve disabilitare il rinfresco automatico, quindi attendere (a polling o a interrupt) la fine dell'ultima trasmissione in corso, e procedere nella gestione dei dati ricevuti come di consueto. Si noti che a questo punto il programma vede nel buffer la situazione dell'ultima serie di transazioni effettuate.
Nella modalità a rinfresco continuo, il rinfresco della rete ETN procede automaticamente all'infinito, finché il programma non lo disabilita. Nella modalità temporanea, dopo un secondo il rinfresco automatico cessa, viene generato l'interrupt di fine trasmissione (se abilitato) e il bit 7 del registro di configurazione standard va a 1.

6.3.6.3. Bit 3, 4, 7

Questi Bit sono riservati e non devono essere utilizzati.

7. IL SISTEMA ETN

7.1. INTRODUZIONE

Nel presente capitolo viene fornita una descrizione sommaria del sistema ETN, per ulteriori informazioni riguardo al sistema ETN riferirsi alle seguenti pubblicazioni Tecnint HTE :

- “Il sistema ETN”
- “Introduzione al sistema ETN”(app. note #2)
- “Calcolo della banda passante di un sistema ETN”(app. note #3)
- “La topologia del sistema ETN”(app. note #4)

7.2. CONFIGURAZIONE HW

Per un corretto funzionamento della rete ETN si raccomanda di seguire le seguenti indicazioni:

- Le resistenze di terminazione vanno inserite alle estremità della linea di trasmissione.
- Configurare la velocità di trasmissione sia sul modulo MASTER TSN-150/PCI che sui moduli SLAVE in modo tale che sia identica tra loro ed adatta alla lunghezza del cavo (vd. par. 6.3.6.1).
- Mediante il cavo bipolare schermato (o la fibra ottica) connettere il modulo TSN-150/PCI ai vari moduli SLAVE e tramite i ponticelli presenti su questi ultimi configurarne correttamente l'indirizzo. Alimentare quindi i moduli SLAVE con una tensione a 24V DC.

7.3. IL PROTOCOLLO ETN

Il colloquio tra il modulo MASTER, in questo caso il modulo TSN-150/PCI, e i vari moduli SLAVE collegati in rete avviene mediante un protocollo a transazioni implementato in modo hardware. Per transazione si intende l'insieme di un messaggio di sollecitazione e della risposta conseguente. Il messaggio di sollecitazione è inviato dal MASTER della rete; la risposta del generico modulo SLAVE segue immediatamente l'arrivo della sollecitazione.
Viene riportato in Tabella 7-1 il formato di un record di transazione tra un modulo MASTER ed un generico modulo SLAVE[1].

BYTE	SIGNIFICATO
0	Indirizzo SLAVE trasmesso
1	Codice di tipo dello SLAVE trasmesso
2	Bits di informazione trasmessi (24 ÷ 31)
3	Bits di informazione trasmessi (16 ÷ 23)
4	Bits di informazione trasmessi (8 ÷ 15)
5	Bits di informazione trasmessi (0 ÷ 7)
6	byte attributi: bit 0: size dati: 0 = 32 bit, 1 = 16 bit; bit 4, 6: selezione linea
7	Non usato
8	Indirizzo dello SLAVE ricevuto
9	Codice di tipo dello SLAVE ricevuto
10	Bits di informazione ricevuti (24 ÷ 31)
11	Bits di informazione ricevuti (16 ÷ 23)
12	Bits di informazione ricevuti (8 ÷ 15)
13	Bits di informazione ricevuti (0 ÷ 7)
14	Indirizzo del modulo MASTER (0x00)
15	Flag di errore (0 = OK, 0xFF = errore)

Tabella 7-1: Struttura di un record di transazione

[1] Il record di transazione è formato da 16 byte.

Viene qui fornito un esempio di utilizzo della rete ETN mediante il modulo TSN-150/PCI. Per maggiori dettagli si rimanda all’appendice dove è presente un programma esemplificativo in linguaggio “C”.

In “Figura 7.1: Ciclo di TX/RX a polling usando un solo buffer.” viene riportato uno schema a blocchi che rappresenta la sequenza di operazioni da effettuare per inizializzare e successivamente trasmettere e ricevere dati da uno o più moduli ETN, con metodo a polling[2] e con modalità di rinfresco tramite accesso al registro Reset Interrupt/Start ETN.

[2] Esiste anche la possibilità ad interrupt al livello configurato come indicato al paragrafo.

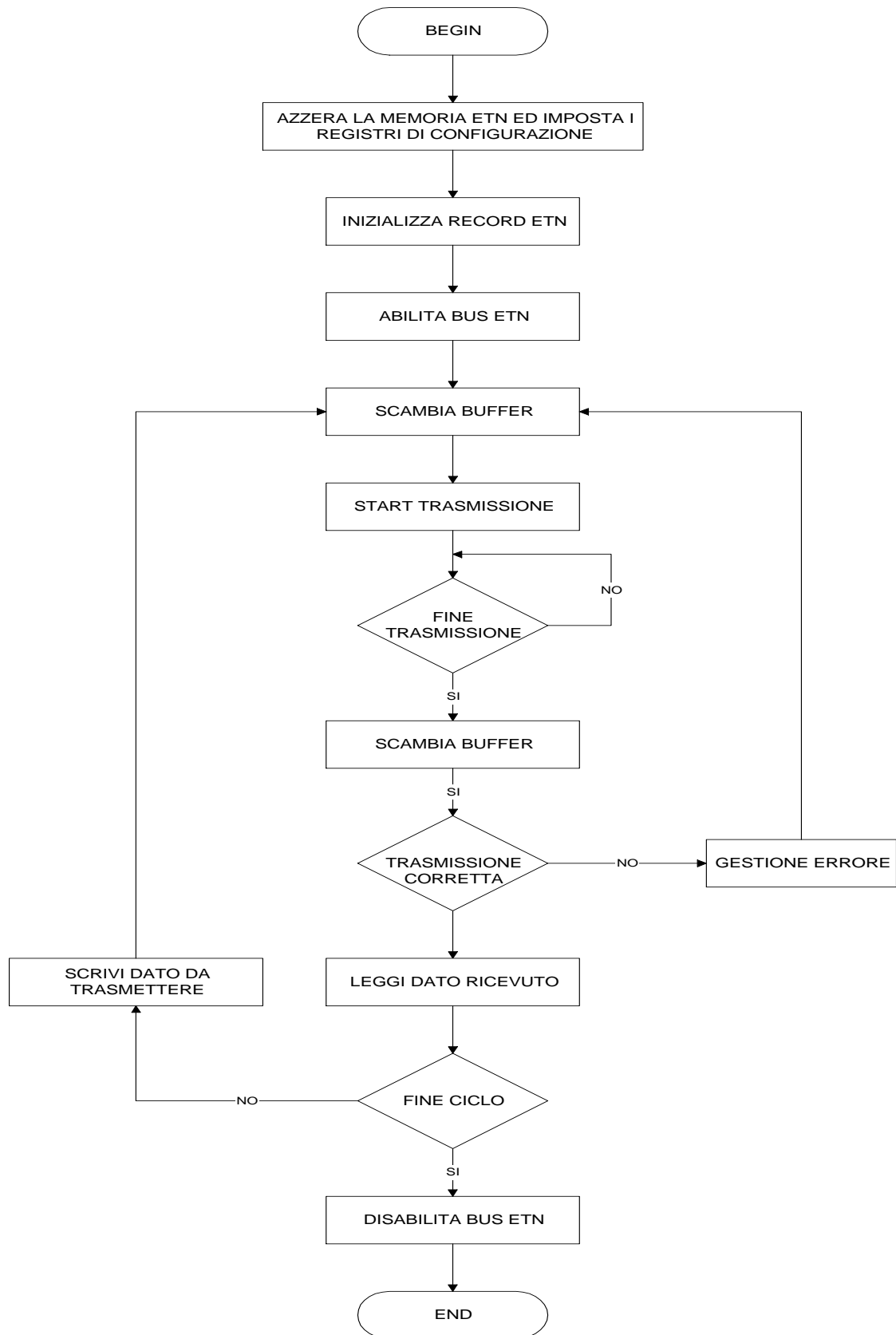



Figura 7.1: Ciclo di TX/RX a polling usando un solo buffer.

Si supponga di voler pilotare le uscite e leggere gli ingressi di un modulo SLAVE TSR-40 a 16 ingressi digitali + 16 uscite digitali a relè. Si assume che il modulo TSR-40 in questione abbia indirizzo 0x01, sia collegato alla linea RS485 n°0, metodo di rinfresco standard. Dopo aver azzerato la memoria ETN e impostato i registri di configurazione[3], si scrivono le locazioni di memoria ETN come specificato nella
Tabella 7-2[4]. Così facendo non si fa altro che scrivere un record di transazione che ha come indirizzo iniziale 0x0010[5].

INDIRIZZO	DATO	COMMENTO
0x0010	0x01	indirizzo modulo slave
0x0011	0x06	tipo modulo (TSR-40) slave
0x0012	0x00	dato hh (valido solo con transazioni a 32bit)
0x0013	0x00	dato hl (valido solo con transazioni a 32bit)
0x0014	0x00	dato lh uscite 8-15
0x0015	0x00	dato ll uscite 0-7
0x0016	0x01	transazione 16 bit, linea 0 RS485
0x0017	0x55	non usato
0x0018	0x55	indirizzo modulo ricevuto
0x0019	0x55	tipo modulo (bit 0-3) + lrc ric. (bit 4-7)
0x001A	0x55	dato hh ricevuto (valido solo con transazioni a 32bit)
0x001B	0x55	dato hl ricevuto (valido solo con transazioni a 32bit)
0x001C	0x55	dato lh ricevuto, stato ingressi 8-15
0x001D	0x55	dato ll ricevuto, stato ingressi 0-7
0x001E	0x55	indirizzo MASTER ricevuto
0x001F	0x55	flag errore (0 =OK; 0xFF =ERRORE)
0x0020	0x00	indirizzo slave invalido = fine records

Tabella 7-2: Esempio di inizializzazione

[3] Vedi paragrafo 6.3.4.
[4] I 32 Kb di memoria ETN sono strutturati in 2 buffer di 16 Kb ciascuno: uno è accessibile al PC, mentre l'altro è in trasmissione (vedi paragrafo **Errore. L'origine riferimento non è stata trovata.**).
[5] Il primo record di ogni transazione è destinato ad usi interni e perciò non deve essere utilizzato per le transazioni. Esso deve essere posto interamente pari a zero per tutti gli indirizzi (0x0000 ÷ 0x000F). Questa azione deve essere effettuata anche per il primo record del secondo buffer (indirizzi 0x4000 ÷ 0x400F).

A questo punto sono stati inizializzati i campi essenziali di un record (indirizzo e tipo modulo), impostati i dati iniziali da trasmettere (dato LH e LL) e riempiti con caratteri 0x55 gli altri campi (sia non usati che usati in ricezione)[6], inoltre è stato delimitato il numero di record nel buffer mediante il carattere di delimitazione 0x00. Per il primo buffer è stato allocato un record, per il secondo nessun record.
Prima di attivare la trasmissione, occorre scambiare i buffer, in modo da mettere in trasmissione il buffer in cui si è scritto il record.
Successivamente è necessario abilitare il bus ETN tramite una scrittura al registro di configurazione, impostando il bit 0 a 1.
A questo punto si attiva la trasmissione sul bus ETN mediante una scrittura nel registro di reset interrupt/start. Il ciclo di trasmissione/ricezione si interrompe quando incontra un record in cui l'indirizzo del modulo SLAVE è pari a 0x00, in questo caso subito dopo aver trasmesso e ricevuto i dati dell'unico record impostato.
Il riconoscimento di fine trasmissione da parte del PC avviene o ricevendo un interrupt (se abilitato), oppure in polling testando il bit 7 di fine trasmissione sul registro di configurazione standard. Quindi si scambiano i buffer per rendere accessibile la transazione effettuata e rendere possibile la lettura dei dati ricevuti.
Nella locazione 0x001F si trova il flag di errore: in caso di errore di trasmissione è impostato a 0xFF, altrimenti ha il valore 0x00.
Se si vuole ottenere un ciclo continuo di trasmissione/ricezione con il modulo SLAVE, devono essere ripetute tutte le azioni successive all'inizializzazione (cioè dallo scambio dei buffer precedente l'abilitazione del bus ETN), chiaramente precedute dalla scrittura dei nuovi dati da trasmettere.

7.3.1. Selezione linea di trasmissione

La selezione del mezzo fisico di trasporto delle transazioni ETN si impostano con i bit 4 e 6 del byte di attributi del record ETN.
I mezzi disponibili sul TSN-150/PCI sono: 2 linee elettriche (linea 0 e linea 1) e una coppia di fibre ottiche. La selezione della linea di trasmissione è indicata dalla seguente tabella.

Selezione linea di trasmissione		
bit 6	**bit 4**	**linea selezionata**
0	0	RS485 linea 0
0	1	RS485 linea 1
1	x	fibra ottica

Tabella 7-3: Selezione della linea di Trasmissione su byte 6 (attributi) del record ETN[7]

[6] Operazione opzionale
[7] Vedi struttura record ETN Tabella 7-1.

8. APPENDICI

8.1. Tempo di ciclo ETN, Tcycle

L'aggiornamento dello stato delle uscite e l'acquisizione dello stato degli ingressi dei vari moduli SLAVE avviene ad una velocità che dipende dalla frequenza selezionata sui vari moduli della rete ossia dalla frequenza di lavoro della rete ETN.
Il tempo totale del ciclo di BUS dipende dal baud-rate della rete, dal numero di bit impiegato dal protocollo ETN, dal numero di slave attivi e dal numero di bit scambiati da ogni slave (16 o 32 in output più 16 o 32 bit in input) per ogni transazione.
Il numero di bit scambiati in una transazione a 32 bit è pari a 77, mentre è pari a 55 in una transazione a 16 bit.

Uno slave che accetti 32 bit in ingresso e restituisca 32 bit in uscita attiva due transazioni da 77+77=**154** bit.
Ad ogni transazione vanno poi aggiunti **22** bit usati dal master, per cui i bit effettivamente utilizzati sono 154+22=**176** per le transazioni a 32bit e 110+22=**132** per transazioni a 16bit.

Il tempo di ciclo del BUS ETN può essere calcolato come sommatoria del tempo impiegato da ogni singolo slave presente nella rete utilizzando :

Tslave(0..n) = ((n_bit_m + n_bit_s) * Tbit) + (n_mas *Tbit)

Tcycle= Tslave(0)+….+Tslave(n)

Dove **Tbit** è il tempo necessario a trasmettere un bit, **n_bit_m** è il numero di bit trasmessi dal master verso lo slave, **n_bit_s** è il numero di bit trasmessi dallo slave al master e **n_mas** è un numero di bit aggiuntivo utilizzati dal master, pari a **22**.

8.1.1. Tempo massimo di ciclo ETN Tcycle_max

il tempo massimo di ciclo può essere calcolato per eccesso considerando tutti gli slave a 32bit ed utilizzando la seguente espressione:

Tcycle_max = (176 * Tbit * N_slaves)

A seguito si riportano le tabelle dei tempi teorici di ciclo, per singolo record e per tutti i 1022 records.

Velocità linea	Tbit	RECORD 16 BIT	record 32 bit
12 Mbit/s	83 ns	10.956 us	14.618 us
6 Mbit/s	167 ns	22.044 us	29.392 us
3 Mbit/s	333 ns	43.956 us	58.608 us
1,5 Mbit/s	667 ns	88.044 us	117.392 us
750 Kbit/s	1.333 us	0.175956 ms	0.234608 ms
375 Kbit/s	3.667 us	0.484044 ms	0.645392 ms
187 Kbit/s	5.348 us	0.705936 ms	0.941248 ms
93,75 Kbit/s	10.667 us	1.408044 ms	1.877392 ms

Tabella 8-1: Tabella tempi di trasferimento singolo record ETN per baud-rate

Velocità linea	Tbit	1022 RECORDS /16 BIT	1022 records / 32 bit
12 Mbit/s	83 ns	11.197 ms	14.939596 ms
6 Mbit/s	167 ns	22.52488 ms	30.038624 ms
3 Mbit/s	333 ns	44.923032 ms	59.897376 ms
1,5 Mbit/s	667 ns	89.980968 ms	119.974624 ms
750 Kbit/s	1.333 us	179.827032 ms	239.769376 ms
375 Kbit/s	3.667 us	494.692968 ms	659.590624 ms
187 Kbit/s	5.348 us	721.466592 ms	961.955456 ms
93,75 Kbit/s	10.667 us	1.4390210 s	1.9186946 s

Tabella 8-2: Tabella di tempi massimi tempi di trasferimento record ETN

8.2. Registri PCI

I registri PCI necessari per leggere le informazioni relative alla base dello spazio di I/O e a quello di memoria sono standard e sono disponibili tramite le funzioni del sistema operativo.

Per identificare la scheda, sono disponibili i registri VENDOR ID, DEVICE ID, SUBVENDOR ID e SUBSYSTEM ID.

I codici VENDOR ID e DEVICE ID sono assegnati dal costruttore del chip di interfaccia PCI e sono fissi: nel caso della TSN150/PCI valgono rispettivamente:

VENDOR ID = 0x10B5
DEVICE ID = 0x9050

I codici di SUBVENDOR ID e SUBSYSTEM ID sono assegnati dal costruttore della scheda e memorizzati da una EEPROM a bordo.

Per la scheda TSN150/PCI sono stati assegnati i seguenti valori:

SUBVENDOR ID = 0x10B5
SUBSYSTEM ID = 0x2277

8.2.1. Classe del device PCI scheda TSN150

Le schede TSN150PCI sono configurate come device PCI di classe **0x02** (**NETWORK)** e **subclass 0x80** (“**other**”).

8.2.1.1. PCI Configuration Space registers

I dispositivi PCI dispongono di un set di registri standard, in cui sono allocate le risorse per accedere al device PCI vero e proprio.
A seguito si riportano alcuni di questi registri:

PCI Configuration Space registers		
NAME	**OFFSET**	**Value/Description**
VENDOR_ID	**0x0000**	**0x9050**
DEVICE_ID	**0x0002**	**0x10B5**
COMMAND	0x0004	
STATUS	0x0006	
REVISION_ID	0x0008	
CLASS_CODE	**0x000A**	**0x0280**
CACHE_LINE_SIZE	0x000C	
MASTER_LATENCY	0x000D	
HEADER_TYPE	0x000E	
BIST	0x000F	
BASE_ADDRESS_0	**0x0010**	PLX registers (Memory space)
BASE_ADDRESS_1	**0x0014**	PLX registers (I/O space)
BASE_ADDRESS_2	**0x0018**	**ISA_MEMCS_BASE**
BASE_ADDRESS_3	**0x001C**	**ISA_IOCS_BASE**
BASE_ADDRESS_4	0x0020	ISA_CSRAM_BASE
BASE_ADDRESS_5	0x0024	ISA_CSROM_BASE
SUBVENDOR_ID	**0x002C**	**0x10B5**
SUBSYSTEM_ID	**0x002E**	**0x2277**
EXPANSION_ROM	0x0030	
INTERRUPT_LINE	**0x003C**	**Interrupt line (IRQ)**
INTERRUPT_PIN	0x003D	Interrupt pin
MIN_GNT	0x003E	
MAX_LAT	0x003F	

Tabella 8-3: Registri configurazione PCI

8.2.1.1.1. Registri PCI di risorse della scheda

I registri necessari per gestire la scheda TSN150 PCI, oltre a quelli per identificare il dispositivo (VENDOR_ID, DEVICE_ID, SUBVENDOR_ID , SUBSYSTEM_ID e CLASS_CODE), sono quelli che definiscono le risorse di I/O, Memoria e Interrupt:

BASE_ADDRESS_0: Questo registro contiene l' indirizzo di memoria fisico assegnato ai registri a 32bit di controllo interni del chip PLX. Occorre notare che con sistemi operativi tipo Windows, Linux o QNX questo indirizzo non può essere utilizzato come letto dal registro: va prima mappato nello spazio di memoria virtuale del processore. Lo spazio reale a disposizione è di 128 byte

BASE_ADDRESS_1: Questo registro contiene l' indirizzo di I/O fisico assegnato ai registri a 32bit di controllo interni del chip PLX. E' in alternativa all' uso dell' accesso tramite BASE_ADDRESS_0.

BASE_ADDRESS_2: Questo registro contiene l' indirizzo di memoria fisico assegnato alla scheda. Occorre notare che con sistemi operativi tipo Windows, Linux o QNX questo indirizzo non può essere utilizzato come letto dal registro: va prima mappato nello spazio di memoria virtuale del processore. Lo spazio reale a disposizione è di 32Kbyte

BASE_ADDRESS_3: Questo registro contiene l' indirizzo di I/O fisico assegnato alla scheda. Occorre notare che con sistemi operativi tipo Windows, Linux o QNX questo indirizzo potrebbe non essere utilizzato come letto dal registro: va prima mappato nello spazio di memoria virtuale del processore. Se la CPU su chi è inserita la TSN150 è di tipo PC x86, la mappatura dell' indirizzo di I/O non dovrebbe essere necessaria, ma occorre avere diritti di accesso all' I/O.
Va inoltre notato che il bit più significativo è sempre settato a 1 ad indicare che lo spazio è di tipo I/O e quindi va mascherato.

INTERRUPT_LINE: In questo registro è contenuto il livello di IRQ assegnato alla scheda.

L' uso delle funzionalità ETN è completamente definito utilizzando le informazioni contenute nei registri **BASE_ADDRESS_2**, **BASE_ADDRESS_3** e **INTERRUPT_LINE**.

8.2.1.1.2. Controllo degli USER LED

La scheda TSN150 PCI è dotata di due LED controllabili da software.
I due led sono pilotati da due pin di I/O direttamente dal chip bridge PCI PLX .

Per utilizzare i led è necessario accedere ad un registro a 32bit interno al chip PLX denominato CONTROL, posto all' offset **0x50** rispetto alla base definita dal registro **PCI BASE_ADDRESS_0.**

Del registro di controllo, l' utente deve modificare solo ed esclusivamente i bit di configurazione dei due pin assegnati ai led USER0 e USER1, non deve modificare le altre impostazioni:

BIT	**Funzione**		
0	Funzione:	0= pin di I/O	**Impostare sempre a 0**
1	Direzione:	0=Ingresso, 1=Uscita	**Impostare sempre a 1**
2	**Stato ingresso/Uscita**		**1= Led ON, 0=Led Off**
3	Funzione:	0= pin di I/O	**Impostare sempre a 0**
4	Direzione:	0=Ingresso, 1=Uscita	**Impostare sempre a 1**
5	**Stato ingresso/Uscita**		**1=Led ON, 0=Led Off**
6..31	**NON MODIFICARE**		

ATTENZIONE: i registri interni al chip PLX sono configurati per controllare i parametri di accesso al BUS PCI e non devono essere modificati dall' utente.

Del registro CONTROL, solo i bit da 0 a 6 possono essere modificati dall' utente per potere controllare i due LED: tutti gli altri bit sono riservati e non vanno modificati.

Modifiche anche accidentali nei registri di configurazione possono causare gravi malfunzionamenti della scheda TSN150PCI e al limite il blocco del bus PCI su cui è inserita.

8.3. Interfaccia RS485

L’ interfaccia seriale RS485 è uno standard di fatto basato su una linea di comunicazione differenziale bilanciata, con una impedenza tipica di **120 ohm.**
La comunicazione e’ di tipo half-duplex ed e’ utilizzata in protocolli one-to-many o master-slave, come il protocollo ETN.
In questo tipo di comunicazione possono essere collegati molti dispositivi sulla stessa linea fisica, ma solo un dispositivo alla volta può trasmettere dati, mentre tutti gli altri dispositivi possono solo ricevere.

La lunghezza massima del collegamento fisico tra i vari dispositivi dipende dalla velocità di comunicazione, dal rapporto segnale disturbo, dalla qualità del cavo e dal numero di punti collegati e si aggira sui **1200m**.

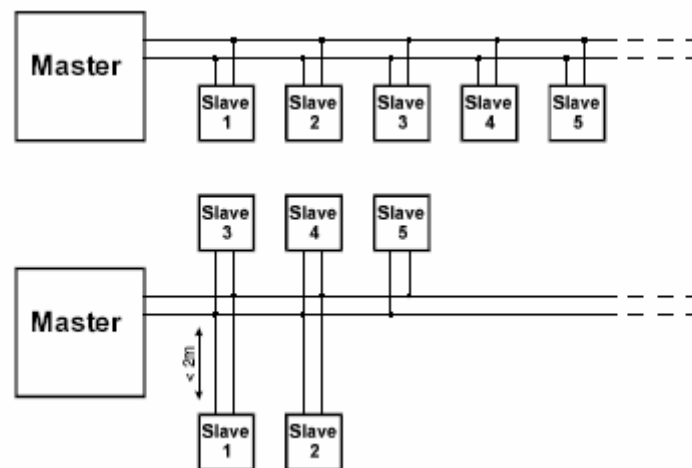
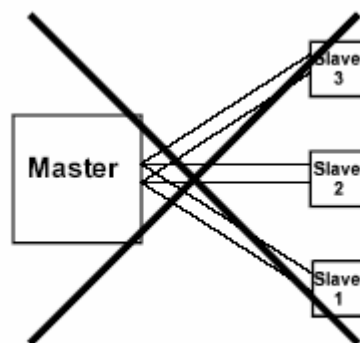
8.3.1. Tipi di cavi

I cavi di collegamento consigliati per avere le migliori prestazioni sono di tipo schermato con impedenza di **120Ohm** (ad esempio, **BELDEN 3079E, BELDEN 9841** oppure **CEAM CPR 6003**, ottimi i cavi utilizzati per collegamenti PROFIBUS).

E’ possibile usare cavi senza particolari caratteristiche, anche non schermati, se la distanza è di qualche metro in ambiente elettricamente poco rumoroso e si utilizzano basse velocità di trasmissine (inferiori a 38K bit/s).

Per distanze comprese tra **15** e **100m** è possibile usare un cavo intrecciato schermato (twistato) senza particolari caratteristiche, mentre per collegamenti oltre i **100m** o ad alte velocità è consigliabile utilizzare cavi specificamente progettati per trasmissione dati RS485.

8.3.2. Tipologia di cablaggio

Il cablaggio della linea di comunicazione deve preferibilmente essere effettuato a catena, evitando configurazioni a stella e limitando le derivazioni a pochi metri (vedi disegni).

Tipologia di cablaggio a catena

Tipologia di cablaggio a stella (da evitare)

Gli estremi della linea devono essere terminati inserendo in parallelo alla linea una resistenza di 120 ohm: la terminazione di linea e' sempre presente sui master e sugli slave ETN e basta configurare il ponticello del primo e dell' ultimo apparato in modo corretto.
Lo schermo del cavo deve essere collegato alla massa comune da entrambi i lati, e collegato a terra almeno da un lato.
E' possibile, in caso di linee particolarmente rumorose, ridurre i disturbi collegando l 'altro lato dello schermo a terra per mezzo di un condensatore da **10nF**.

ATTENZIONE:
Evitare di inserire terminazioni nel mezzo della linea! Terminazioni mal posizionate o terminazioni aggiuntive rispetto alle due necessarie possono causare gravi malfunzionamenti nella comunicazione tra gli apparati.

8.4. Sorgenti di Esempio

8.4.1. Esempio accesso in modalità I/O

```
/*--------------------------------------
 Esempio
--------------------------------------*/

#include <tsn150.h>

/* BEGIN */
void main(void)
{
    unsigned char *c,record[16];

    /* AZZERA ED IMPOSTA... */
    etn_cold_init();
    /* INIZIALIZZA RECORD */
    for ( c = record; c < & record[16]; *c++ = 0 );
    record[0] = 0x01;    // indirizzo modulo TSR-40
    record[1] = 0x06;    // tipo - TSR-40
    record[6] = 0x01;    // linea 0 RS485, 16 bit
    etn_put_rec(0,record);
    /* ABILITA BUS */
    etn_enable();
    while ( condizione ciclo )
    {
        /* SCAMBIA BUFFER */
        etn_change_txbuff();
        /* START TRASMISSIONE */
        etn_start_transm();
        /* ATTESA FINE TRASMISSIONE */
        while ( !etn_test_end() );
        /* SCAMBIA BUFFER */
        etn_change_txbuff();
        /* TRASMISSIONE CORRETTA? */
        if ( etn_test_err(0) )
        {
            gestisci_errore();
            continue;
        }
        /* LETTURA DATO RICEVUTO */
        datoRx = etn_get_16value(0);
        /* SCRITTURA DATO DA TRASMETTERE */
        datoTx = ~datoRx;
        etn_put_16value(0,datoTx);
    }
    /* DISABILITA BUS */
    etn_disable();
}
/* END */
```

File: TSN150.H

```
/*    TSN150 Driver Interface
    Version 1.0 - 18/10/96 - Pma
*/

                           /* transmitted value */
#define    txaddr       0  /*  slave module address */
#define    txtipo       1  /*  slave module type */
#define    txhh         2  /*  data hh (32 bit only) */
#define    txhl         3  /*  data hl (32 bit only) */
#define    txlh         4  /*  data lh */
#define    txll         5  /*  data ll */
#define    txattr       6  /*  flags */
#define    txfrge       7  /*  unused */
                           /* received value */
#define    rxaddr       8  /*  received slave address */
#define    rxtipo       9  /*  received slave type */
#define    rxhh        10  /*  data hh (32 bit only) */
#define    rxhl        11  /*  data hl (32 bit only) */
#define    rxlh        12  /*  data lh */
#define    rxll        13  /*  data ll */
#define    rxmaster    14  /*  received master address */
#define    rxerr       15  /*  error flag */

#define etnreclen  16  /*  ETN record length */

void etn_set_base(unsigned int base);
void etn_irq_level(unsigned int level);
unsigned char etn_get_configreg(void);
unsigned char etn_get_extconfigreg(void);
void etn_start_transm(void);
void etn_enable(void);
void etn_disable(void);
void etn_int_enable(void (*user)(void));
void etn_int_disable(void);
void etn_set_txbuff0(void);
void etn_set_txbuff1(void);
void etn_change_txbuff(void);
int etn_get_txbuff(void);
int etn_test_end(void);
void outbyte150(unsigned int abs_addr, unsigned char value );
void outword150(unsigned int abs_addr, unsigned int value );
void outlong150( unsigned int abs_addr,unsigned long value );
unsigned char inbyte150( unsigned int abs_addr);
unsigned int inword150( unsigned int abs_addr );
unsigned long inlong150( unsigned int abs_addr );
unsigned int etn_fetch_rec(unsigned int recnum);
void etn_get_rec(unsigned int recnum, unsigned char *ptr);
void etn_put_rec(unsigned int recnum, unsigned char *ptr);
void etn_clear_rec(unsigned int recnum);
unsigned char etn_test_err(unsigned int recnum);
unsigned long etn_get_32value(unsigned int recnum);
void etn_put_32value(unsigned int recnum, unsigned long value);
unsigned int etn_get_16value(unsigned int recnum);
void etn_put_16value(unsigned int recnum, unsigned int value);
void etn_warm_init(void);
void etn_cold_init(void);
void etn_swsbrate_enable(void);
void etn_swsbrate_disable(void);
unsigned char etn_set_brate(unsigned char brate_value);
void etn_channelsel_enable(void);
void etn_channelsel_disable(void);
void etn_autorefwt_enable(void);
void etn_autorefwt_disable(void);
void etn_autoref_enable(void);
void etn_autoref_disable(void);
unsigned int etn_ramptr_save(void);
void etn_ramptr_restore(unsigned int ramETNptr);
unsigned int etn_count_records(void);
```

File: TSN150.C

```
/***************************************************************
 * (C) Copyright 1990, 1996 by TECNINT HTE
 * Title: TSN 150 driver
 * Rev.    1.0 -- 18/09/96 -- Pma
 ***************************************************************/
#include <dos.h>
#include "tsn150.h"

/********************   ADDRESS SPACE   ************************
 *    xxx = TSN-150 I/O base address
 *    $xxx0 - low pointer ETN ram address register ( R/W )
 *    $xxx1 - high pointer ETN ram address register ( R/W )
 *    $xxx2 - data ETN ram register ( R/W )
 *    $xxx3 - configuration register ( R/W )
 *    $xxx4 - start trasmission & reset interrupt pending register ( R/W )
 *    $xxx5 - extended configuration register ( R/W )
 *
 ****************************************************************/

/*********   CONFIGURATION REGISTER - BIT SPECIFICATION   *********
 *
 *     bit  -------  0  -------           -------  1  -------
 *      0   ETN disable                   ETN enable
 *      1   interrupt disable             interrupt enable
 *      2   txbuff 0                      txbuff 1
 *      3        reserved                  reserved
 *      4        reserved                  reserved
 *      5        reserved                  reserved
 *      6        reserved                  reserved
 *      7   ETN tx in progress            ETN tx is over
 *
 ****************************************************************/

/******   EXTENDED CONFIGURATION REGISTER - BIT SPECIFICATION   ******
 *
 *     bit
 *      0    -\                                    /- 000 max baud rate
 *      1     |--- ETN baud rate selection =---|
 *      2    -/                                    \- 111 min baud rate
 *      3    enable ETN baud rate selection via ext. config. register
 *      4    enable ETN channel selection via ETN record attribute byte
 *      5    enable automatic ETN refresh with timeout (about 1 sec.)
 *      6    enable automatic ETN refresh continuous mode
 *      7    reseved
 *
 ****************************************************************/

/***************************************************************
 * TSN 150 Mask
 ***************************************************************/
#define etn_enable_msk        0x01
#define etn_irq_enable_msk    0x02
#define etn_txbuff_msk        0x04
#define etn_irq_event_msk     0x80
#define etn_autoref_msk       0x40
#define etn_autorefwt_msk     0x20
#define etn_chselen_msk       0x10
#define etn_swsbrate_msk      0x08
#define etn_bratebit_msk      0x07

/***************************************************************
 * Costants
 ***************************************************************/
#define DEFAULT_IO_BASE 0x300
#define DEFAULT_IRQ_LEV 5
#define DEFAULT_IRQ_VEC 0x0D

/***************************************************************
 * Private variables
 ***************************************************************/
static unsigned int addr_baseL = DEFAULT_IO_BASE;
static unsigned int addr_baseH = DEFAULT_IO_BASE+1;
static unsigned int addr_dato = DEFAULT_IO_BASE+2;
static unsigned int addr_config = DEFAULT_IO_BASE+3;
```

```
static unsigned int addr_clrint = DEFAULT_IO_BASE+4;
static unsigned int addr_extconfig = DEFAULT_IO_BASE+5;
static void (*user_isr)(void) = 0;
static void interrupt (*old_vec_ptr)(void);
static int irq_level = DEFAULT_IRQ_LEV;
static int irq_vector = DEFAULT_IRQ_VEC;

/*****************************************************************
 * Private prototypes
 *****************************************************************/
void irqn_enable (int en_irq);
void irqn_disable (int dis_irq);
void interrupt irq150(void);

/*  Public ======*/

/***********************************************************
 * TSN150  etn_set_base
 *
 * input    TSN150 base address
 * output    none
 ***********************************************************/
void etn_set_base(unsigned int base)
{
    addr_baseL     = base;
    addr_baseH  = base+1;
    addr_dato   = base+2;
    addr_config = base+3;
    addr_clrint = base+4;
    addr_extconfig = base+5;
}

/***********************************************************
 * TSN150  etn_irq_level
 *
 * input    TSN150 IRQ level
 * output    none
 ***********************************************************/
void etn_irq_level(unsigned int level)
{
    static char valid[] = {0,0,0,1,1,1,0,1};

    level &= 7;
    if ( valid[level] )
    {
        irq_level = level;
        irq_vector = level + 8;
    }
}

/***********************************************************
 * TSN150  etn_get_configreg
 *
 * input    none
 * output    - a byte with the content of the
 *             configuration register (refer to the
 *             configuration register map to have bits
 *             meaning)
 ***********************************************************/
unsigned char etn_get_configreg(void)
{
    return ( inportb(addr_config) );
}

/***********************************************************
 * TSN150  etn_get_extconfigreg
 * input    none
 * output    - a byte with the content of the
 *      extended configuration register (refer to the
 *      extended configuration register map to have bits
 *             meaning)
 ***********************************************************/
unsigned char etn_get_extconfigreg(void)
{
    return ( inportb(addr_extconfig) );
}
```

```
/*********************************************************
 * TSN150  etn_start_transm
 *
 * input    none
 * output    none
 * effect    ETN start the transmission of the
 *          currently active txbuff
 *********************************************************/
void etn_start_transm(void)
{
    outportb(addr_clrint,0x11); /* any access is enought */
}

/*********************************************************
 * TSN150   etn_enable
 * input    none
 * output    none
 * effect    ETN transmission is enabled, but not
 *          started
 *********************************************************/
void etn_enable(void)
{
    outportb( addr_config,(etn_enable_msk | inportb(addr_config)) );
}

/*********************************************************
 * TSN150  etn_disable
 * input    none
 * output    none
 * effect    ETN transmission is disabled
 *********************************************************/
void etn_disable(void)
{
    outportb( addr_config,(~etn_enable_msk & inportb(addr_config)) );
}

/*********************************************************
 * TSN150  etn_int_enable
 * input    user interrupt routine
 * output    none
 * effect    TSN150 is enabled to assert a interrupt
 *          on the end of ETN transmission
 *********************************************************/
void etn_int_enable(void (*user)(void))
{
    user_isr = user;
    disable();
    /* save old handler */
    old_vec_ptr = getvect(irq_vector);
    /* set new handler */
    setvect(irq_vector,irq150);
    /* enable IRQ level */
    irqn_enable(irq_level);
    enable();
    /* enable TSN-150 interrupt */
    outportb( addr_config,(etn_irq_enable_msk | inportb(addr_config)) );
}

/*********************************************************
 * TSN150  etn_int_disable
 * input    none
 * output   none
 * effect   interrupt asserting on the end of ETNis disabled
 *********************************************************/
void etn_int_disable(void)
{
    /* disable TSN-150 interrupt */
    outportb( addr_config,(~etn_irq_enable_msk & inportb(addr_config)) );
    disable();
    setvect(irq_vector,old_vec_ptr);   /* restore old handler */
    /* disable IRQ level */
    irqn_disable(irq_level);
    enable();
    user_isr = 0;
}
```

```
/*********************************************************
  * TSN150  etn_set_txbuff0
 *
 * input    none
 * output    none
 * effect    txbuff 0 became the available txbuff and
 *          txbuff 1 become the transmitting buffer
 * NOTE!    No other ETN management is done inside the
 *          routine; it should be done outside
 *
 **********************************************************/
void etn_set_txbuff0(void)
{
    outportb( addr_config, (~etn_txbuff_msk  & inportb(addr_config)) );
}

/*********************************************************
 * TSN150  etn_set_txbuff1
 *
 * input    none
 * output    none
 * effect    txbuff 1 became the available txbuff and
 *          txbuff 0 become the transmitting buffer
 **********************************************************/
void etn_set_txbuff1(void)
{
    outportb( addr_config,(etn_txbuff_msk | inportb(addr_config)) );
}

/*********************************************************
 * TSN150  etn_change_txbuff
 *
 * input    none
 * output    none
 * effect    the currently active txbuff is switched
 *          with the currently available txbuff
 **********************************************************/
void etn_change_txbuff(void)
{
    outportb( addr_config,(etn_txbuff_msk ^ inportb(addr_config)) );
}

/*********************************************************
 * TSN150  etn_get_txbuff
 *
 * input    none
 * output    - 0 if the currently available txbuff is 0
 *               1 if the currently available
 *          txbuff is 1
 **********************************************************/
int etn_get_txbuff(void)
{
    return( (inportb( addr_config) & etn_txbuff_msk)>>2 );
}

/*********************************************************
 * TSN150  etn_test_end
 * input    none
 * output    - different from 0 (true) if the
 *              transmission is finished
 * effect    none
 **********************************************************/
int  etn_test_end(void)
{
    return (etn_get_configreg() & etn_irq_event_msk);
}
```

```
/*********************************************************
 * TSN150  outbyte150
 * input     - ETN absolute address
 *           - byte value to write at the ETN address;
 * output    none
 * effect    write a byte at specified address
 *          on tsn150
 *********************************************************/
void outbyte150(unsigned int abs_addr, unsigned char value )
{
    outportb(addr_baseL,abs_addr);            /* set pointer */
    outportb(addr_baseH,abs_addr = abs_addr >> 8);
    outportb(addr_dato,value);
}

/********************************************************
 * TSN150  outword150
 *
 * input     - ETN absolute address
 *           - word value to write at the ETN address;
 * output    none
 * effect    A word is writed on tsn150 at the
 *          specified address
 *          The bytes are writed in the same order of
 *          the source data
 *********************************************************/
void outword150(unsigned int abs_addr, unsigned int value )
{
    outportb(addr_baseL,abs_addr);            /* set pointer */
    outportb(addr_baseH,abs_addr >> 8);

    outportb(addr_dato,value >> 8);
    outportb(addr_dato,value);
}

/*********************************************************
 * TSN150  outlong150
 *
 * input     - ETN absolute address
 *           - word value to write at the ETN address;
 * output    none
 * effect    A long is writed on tsn150 at the
 *          specified address
 *          The bytes are writed in the same order of
 *          the source data
 *********************************************************/
void outlong150( unsigned int abs_addr,unsigned long value )
{
    long unsigned int temp;
    temp = value;

    outportb(addr_baseL,abs_addr);            /* set pointer */
    outportb(addr_baseH,abs_addr >> 8);

    outportb(addr_dato,temp >> 24);                  /* out data */
    outportb(addr_dato,temp >> 16);
    outportb(addr_dato,temp >> 8);
    outportb(addr_dato,temp);
}

/*********************************************************
 * TSN150  inbyte150
 *
 * input     - ETN absolute address
 * output    - byte value picked from the ETN
 *********************************************************/
unsigned char inbyte150( unsigned int abs_addr)
{
    outportb(addr_baseL,abs_addr);            /* set pointer */
    outportb(addr_baseH,abs_addr = abs_addr >> 8);

    return ( inportb(addr_dato) );
}
```

```
/***********************************************************
 * TSN150  inword150
 *
 * input     - ETN absolute address
 * output    - word value picked from the ETN; the
 *          bytes order is the same of the
 *          source data
 ***********************************************************/
unsigned int inword150( unsigned int abs_addr )
{
    register int tmp;

    outportb(addr_baseL,abs_addr);            /* set pointer */
    outportb(addr_baseH,abs_addr >> 8);

    tmp = inportb(addr_dato);
    tmp = ( tmp << 8 ) | (inportb(addr_dato));
    return ( tmp ) ;
}

/***********************************************************
 * TSN150  inlong150
 *
 * input     - ETN absolute address
 * output    - long value picked from the ETN; the
 *            bytes order is the same of the
 *            source data
 ***********************************************************/
unsigned long inlong150( unsigned int abs_addr )
{
    register unsigned long tmp;
    outportb(addr_baseL,abs_addr);            /* set pointer */
    outportb(addr_baseH,abs_addr >> 8);

    tmp = 0;
    tmp = inportb(addr_dato);
    tmp = ( tmp << 8 ) | (inportb(addr_dato));
    tmp = ( tmp << 8 ) | (inportb(addr_dato));
    tmp = ( tmp << 8 ) | (inportb(addr_dato));

    return ( tmp );
}

/***********************************************************
 *
 * TSN150 etn_fetch_rec
 *
 * input    - ETN record number
 * output    - ETN internal address of the record on
 *            the available buffer
 *
 ***********************************************************/
unsigned int etn_fetch_rec(unsigned int recnum)
{
        return(( recnum + 1 ) << 4 );
}
```

```
/*********************************************************
 * TSN150 etn_get_rec
 * input    - ETN source record number
 *          - address of the host record target buffer
 * output    none
 * effect    The whole ETN source record of the
 *          available txbuff is write
 *          inside the host record target buffer
 *********************************************************/
void etn_get_rec(unsigned int recnum, unsigned char *ptr)
{
    unsigned int rec_etn_base, i;
    rec_etn_base = etn_fetch_rec(recnum);
    outportb(addr_baseL  ,rec_etn_base);            /* set pointer */
    outportb(addr_baseH ,rec_etn_base >> 8);
    for ( i=0 ; i< etnreclen ; i++ )
    {
        *ptr = inportb( addr_dato );                  /* out data */
        ptr ++;
    }
}

/*********************************************************
 * TSN150 etn_put_rec
 * input    - ETN target record number
 *          - address of the host record buffer
 * output    none
 * effect    16 bytes of the host record buffer content
 *          is writed inside the whole ETN target
 *          record of the available txbuff
 *********************************************************/
void etn_put_rec(unsigned int recnum, unsigned char *ptr)
{
    unsigned int rec_etn_base, i;
    rec_etn_base = etn_fetch_rec(recnum);
    outportb(addr_baseL  ,rec_etn_base);            /* set pointer */
    outportb(addr_baseH ,rec_etn_base >> 8);
    for ( i=0 ; i < etnreclen ; i++ )
        outportb(addr_dato,*ptr++);                  /* out data */
}

/*********************************************************
 * TSN150 etn_clear_rec
 * input    - ETN target record number
 * output    none
 * effect    The ETN target record of the currently
 *            abvailabl txbuff is filled with zeros
 *********************************************************/
void etn_clear_rec(unsigned int recnum)
{
    unsigned int rec_etn_base, i;

    rec_etn_base = etn_fetch_rec(recnum);

    outportb(addr_baseL  ,rec_etn_base);            /* set pointer */
    outportb(addr_baseH ,rec_etn_base >> 8);

    for ( i=0 ; i < etnreclen ; i++ )
        outportb(addr_dato , 0 );                 /* out data */
}

/*********************************************************
 * TSN150  etn_test_err
 * input    - ETN target record number
 * output    - error byte of the record (rxerr)
 *            0 => ok, 0xff => error
 *********************************************************/
unsigned char  etn_test_err(unsigned int recnum)
{
    unsigned int rec_etn_base;

    rec_etn_base = etn_fetch_rec(recnum);
    return ( inbyte150 (rec_etn_base + rxerr ) );
}
```

```
/************************************************************
 * TSN150 etn_get_32value
 *
 * input     - ETN source record number
 * output    - value (32 bit) picked from the source ETN
 *             record
 * effect    The value is picked from the identified
 *          record of the currently available txbuff
 ************************************************************/
unsigned long etn_get_32value(unsigned int recnum)
{
    unsigned int rec_etn_base;

    rec_etn_base = etn_fetch_rec(recnum);
    return ( inlong150 (rec_etn_base + rxhh));      /* get value */
}

/************************************************************
 * TSN150 etn_put_32value
 *
 * input     - destination record number
 *           - value (32 bit) to put into the destination
 *             record
 * output    none
 * effect    The value is writed inside the identified
 *          record of the currently available txbuff
 ************************************************************/
void etn_put_32value(unsigned int recnum, unsigned long value)
{
    unsigned int rec_etn_base;

    rec_etn_base = etn_fetch_rec(recnum);
    outlong150(rec_etn_base + txhh , value );      /* set value */
}

/************************************************************
 * TSN150 etn_get_16value
 *
 * input     - ETN source record number
 * output    - value (16 bit) picked from the source ETN
 *             record
 * effect    The value is picked from the identified
 *          record of the currently available txbuff
 ************************************************************/
unsigned int etn_get_16value(unsigned int recnum)
{
    unsigned int rec_etn_base;

    rec_etn_base = etn_fetch_rec(recnum);
    return ( inword150 (rec_etn_base + rxlh));      /* get value */
}

/************************************************************
 * TSN150 etn_put_16value
 *
 * input     - destination record number
 *           - value (16 bit) to put into the destination
 *             record
 * output    none
 * effect    The value is writed inside the identified
 *          record of the currently available txbuff
 ************************************************************/
void etn_put_16value(unsigned int recnum, unsigned int value)
{
    unsigned int rec_etn_base;
    rec_etn_base = etn_fetch_rec(recnum);
    outword150(rec_etn_base + txlh , value );      /* set value */
}
```

```
/*********************************************************
 *
 * TSN150 etn_warm_init
 *
 * input    none
 * output    none
 * effect    after this routine:
 *        - ETN is disabled
 *        - ETN interrupt is disabled
 *        - txbuff0 (phase 0) is available to CPU
 *        - All the txbuffs records are unchanged
 *        - ETN is inactive, no transmission operation executed inside
 *      - disable automatic ETN refresh with timeout
 *      - disable automatic ETN refresh continuous mode
 *
 *********************************************************/
void etn_warm_init(void)
{
    etn_disable();
    etn_int_disable();
    etn_swsbrate_disable();
    etn_channelsel_disable();
    etn_autorefwt_disable();
    etn_autoref_disable();
    etn_set_txbuff0();
}

/*********************************************************
 * TSN150 etn_cold_init
 *
 * input    none
 * output    none
 * effect    after this routine:
 *        - ETN NET and IRQ are disabled
 *        - PHASE 0 is active
 *        - All the txbuffs records are filled with zeros
 *        - ETN is inactive, no transmission operation executed inside
 *      - disable automatic ETN refresh with timeout
 *      - disable automatic ETN refresh continuous mode
 *
 *********************************************************/
void etn_cold_init(void)
{
    unsigned int i;

    etn_disable();
    etn_int_disable();
    etn_set_txbuff0();
    etn_swsbrate_disable();
    etn_channelsel_disable();
    etn_autorefwt_disable();
    etn_autoref_disable();
    for ( i = 0 ; i < 1022 ; i++ )
        etn_clear_rec( i );
    etn_set_txbuff1();
    for ( i = 0 ; i < 1022 ; i++ )
        etn_clear_rec( i );
    etn_set_txbuff0();
}

/*********************************************************
 * TSN150 etn_swsbrate_enable()
 *
 * input
 * output
 * effect: enable ETN baud rate selection via software using
 *            extended configuration register
 *
 *********************************************************/
void etn_swsbrate_enable(void)
{
    outportb( addr_extconfig,(etn_swsbrate_msk | inportb(addr_extconfig)) );
}
```

```
/*********************************************************
 * TSN150 etn_swsbrate_disable()
 * input
 * output
 * effect: disable ETN baud rate selection via software
 *********************************************************/
void etn_swsbrate_disable(void)
{
    outportb( addr_extconfig,(~etn_swsbrate_msk & inportb(addr_extconfig)) );
}

/*********************************************************
 * TSN150 etn_set_brate
 *
 * input:         TSN-150      TSN-150/12M
 *        --------------------------------
 *        0x00 = 6     Mb/s | 12    Mb/s
 *          0x01 = 6     Mb/s | 6     Mb/s
 *          0x02 = 3     Mb/s | 3     Mb/s
 *          0x03 = 1.5   Mb/s | 1.5   Mb/s
 *          0x04 = 750   Kb/s | 750   Kb/s
 *          0x05 = 375   Kb/s | 375   Kb/s
 *          0x06 = 187   Kb/s | 187   Kb/s
 *          0x07 = 93.75 Kb/s | 93.75 Kb/s
 *
 * output: 0 => ok, 0xff => error: baud rate value invalid
 *          0xfe => error: baud rate selection via SW is disable
 * effect: Set ETN baud rate via extended configuration register
  *********************************************************/
unsigned char etn_set_brate(unsigned char brate_value)
{
    unsigned char extconfreg;
    if ( brate_value & ~etn_bratebit_msk )
        return 0xff;
    if ( !(etn_swsbrate_msk | (extconfreg = inportb(addr_extconfig))) )
        return 0xfe;
    outportb( addr_extconfig, extconfreg | brate_value);
    return 0;
}

/*********************************************************
 * TSN150 etn_channelsel_enable()
 * input
 * output
 * effect: enable ETN channel selecvion using attribute byte
 *          of ETN record.
 *********************************************************/
void etn_channelsel_enable(void)
{
    outportb( addr_extconfig,(etn_chselen_msk | inportb(addr_extconfig)) );
}

/*********************************************************
 * TSN150 etn_channelsel_disable()
 * input
 * output
 * effect: disable ETN channel selection using attribute byte
 *          of ETN record.
 *********************************************************/
void etn_channelsel_disable(void)
{
    outportb( addr_extconfig,(~etn_chselen_msk & inportb(addr_extconfig)) );
}

/*********************************************************
 * TSN150 etn_autorefwt_enable()
 *
 * input
 * output
 * effect: enable auto refresh ETN with timeout
 *********************************************************/
void etn_autorefwt_enable(void)
{
    outportb( addr_extconfig,(etn_autorefwt_msk | inportb(addr_extconfig)) );
}
```

```
/*********************************************************
 * TSN150 etn_autorefwt_disable()
 *
 * input
 * output
 * effect: disable auto refresh ETN with timeout
 *
 *********************************************************/
void etn_autorefwt_disable(void)
{
    outportb( addr_extconfig,(~etn_autorefwt_msk & inportb(addr_extconfig)) );
}

/*********************************************************
 * TSN150 etn_autoref_enable()
 *
 * input
 * output
 * effect: enable auto refresh ETN (continuous mode)
 *
 *********************************************************/
void etn_autoref_enable(void)
{
    outportb( addr_extconfig,(etn_autoref_msk | inportb(addr_extconfig)) );
}

/*********************************************************
 * TSN150 etn_autoref_disable()
 *
 * input
 * output
 * effect: disable auto refresh ETN (continuous mode)
 *
 *********************************************************/
void etn_autoref_disable(void)
{
    outportb( addr_extconfig,(~etn_autoref_msk & inportb(addr_extconfig)) );
}

/*********************************************************
 * TSN150 etn_ramptr_save();
 *
 * input
 * output
 * effect: read
 *
 *********************************************************/
unsigned int etn_ramptr_save(void)
{
    unsigned int ramETNptr;

    ramETNptr = inportb(addr_baseH);
    ramETNptr = ramETNptr << 8;
    return ( ramETNptr|inportb(addr_baseL));
}

/*********************************************************
 * TSN150 etn_ramptr_restore();
 *
 * input
 * output
 * effect: restore ETN ram address registers
 *
 *********************************************************/
void etn_ramptr_restore(unsigned int ramETNptr)
{
    outportb(addr_baseL ,ramETNptr);
    outportb(addr_baseH ,ramETNptr >> 8);
}
```

```
/**********************************************************
 * TSN150 etn_count_records();
 *
 * input
 * output: number of valid records
 *
 **********************************************************/
unsigned int etn_count_records(void)
{
    unsigned int r;

    for ( r = 0; inbyte150(etn_fetch_rec(r)) && r < 1023; r++ );
    return r;
}


/*  Private
    =======*/

/* enable PIC interrupt level */
void irqn_enable (int en_irq)
{
    unsigned char imr,mask=0x1;

    mask <<= en_irq;
    mask = ~mask;
    imr = inportb(0x21);    // Read IMR register of master 8259
    imr &= mask;
    outportb (0x21,imr);    // enable interrupt number en_irq
}

/* disable PIC interrupt level */
void irqn_disable (int dis_irq)
{
    unsigned char imr,mask=0x1;



    mask <<= dis_irq;
    imr = inportb(0x21);    // Read IMR register of master 8259
    imr |= mask;
    outportb (0x21,imr);    // disable interrupt number dis_irq
}


/* Interrupt Service Routine */
void interrupt irq150(void)
{
    if ( etn_irq_event_msk & inportb(addr_config) )
    {
        /* 8259: EOI for specific level */
        outportb(0x20, 0x60 | (irq_level & 0x07));
        if ( user_isr )
        {
            user_isr();
        }
    }
```

8.4.2. Esempio funzioni accesso modalità I/O e PCI

Il seguente sorgente può essere utilizzato come parte di un driver WDM in windows per accedere alle funzioni della scheda. Il driver WDM non viene qui descritto: fare riferimento alla documentazione Windows.

```
/*!
$Revision: $
//**************************************************************************
//   Software by  Tecnint SrL, VERONA/MERATE(LC) ITALY
//--------------------------------------------------------------------------
// Original Writer: sma
//--------------------------------------------------------------------------
// PRODUCT IDENTIFICATION : TSN150 DRIVER FOR WINDOWS OS
//--------------------------------------------------------------------------
// Module: TSN150.H
//   TSN-150 Low level interface
//
// Compiler: MSVC
// Debugger: MS
//--------------------------------------------------------------------------
// Copyright (c)  Tecnint SrL.
//
// Redistribution and use in source and binary forms, with or without
// modification, are not permitted without a written permission of
// Tecnint SrL.
// 1. Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
// 3. All advertising materials mentioning features or use of this software
//    must display the following acknowledgement:
//    This product includes software developed by the Tecnint SrL
// 4. The name of the Tecnint SrL cannot be used to endorse or promote
//    products derived from this software without specific prior written
//    permission.
//
// THIS SOFTWARE IS PROVIDED BY THE TECNINT SRL  ``AS IS'' AND ANY EXPRESS
// OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
// DISCLAIMED. IN NO EVENT SHALL TECNINT SRL BE LIABLE FOR ANY DIRECT,
// INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
// HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
// STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
// IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//
//**************************************************************************
//                                      VERSION CONTROL HISTORY
//--------------------------------------------------------------------------
$Log: $
//**************************************************************************
*/
#ifndef _TSN150IO_H_
#define _TSN150IO_H_

#ifdef __cplusplus
extern "C" {
#endif
               //[------------------------------------------]
               //[ LOCAL CONDITIONAL COMPILE SWITCHES       ]
               //[------------------------------------------]

#undef USE_PCI_BYTE_ACCESS   /* Undefine to use LONG data access */

       /*-----------ENABLED OPTIONS --------------*/

typedef unsigned long   DWORD;
typedef unsigned char   BYTE;

#define out8(port, data)      WRITE_PORT_UCHAR ((port), (data))
#define in8(port)             READ_PORT_UCHAR((port))
```

```
/*
 ***************************************************************************
 *   PCI Configuration Space Registers
 * TSN150PCI, PCI\VEN_10B5&DEV_9050&SUBSYS_905010B
 ***************************************************************************
 */                                       // EEPROM PLX-DEF VALUE
#define PCI_CS_VENDOR_ID         (0x00) //  0      9050
#define PCI_CS_DEVICE_ID         (0x02) //  2      10B5
#define PCI_CS_COMMAND           (0x04) //
#define PCI_CS_STATUS            (0x06) //
#define PCI_CS_REVISION_ID       (0x08) //  4      000x (class code: rev. ID NOT downlodable)
#define PCI_CS_CLASS_CODE        (0x0A) //  6      0680 (class-code)
#define PCI_CS_CACHE_LINE_SIZE   (0x0c) //
#define PCI_CS_MASTER_LATENCY    (0x0d) //
#define PCI_CS_HEADER_TYPE       (0x0e) //
#define PCI_CS_BIST              (0x0f) //
#define PCI_CS_BASE_ADDRESS_0    (0x10) //
#define PCI_CS_BASE_ADDRESS_1    (0x14) //
#define PCI_CS_BASE_ADDRESS_2    (0x18) //          0 ISA_MEMCS_BASE
#define PCI_CS_BASE_ADDRESS_3    (0x1c) //          1 ISA_IOCS_BASE
#define PCI_CS_BASE_ADDRESS_4    (0x20) //          2 ISA_CSRAM_BASE
#define PCI_CS_BASE_ADDRESS_5    (0x24) //          3 ISA_CSROM_BASE
#define PCI_CS_SUBVENDOR_ID       (0x2C) //  A      10B5
#define PCI_CS_SUBSYSTEM_ID      (0x2E) //  8      9050-1
#define PCI_CS_EXPANSION_ROM     (0x30) //
#define PCI_CS_INTERRUPT_LINE    (0x3c) //  E      Int. Pin
#define PCI_CS_INTERRUPT_PIN     (0x3d) //
#define PCI_CS_MIN_GNT           (0x3e) //
#define PCI_CS_MAX_LAT           (0x3f) //

#define PCI_CTRL_REG            (0x50) // PLX EEPROM / control register

#define PLX_VENDOR_ID        (0x10B5)       // PLX vendor id
#define PLX_DEVICE_ID        (0x9050)       // PLX 9050 Chip Id
#define PLX_SUBVENDOR_ID     (0x10B5)       // PLX sub-vendor id
#define PLX_SUBSYSTEM_ID     (0x9050)       // PLX subsystem Id (0x9050-1)

/*
 * Tecnint Device identification: The SUBVENDOR used is the PLX ID
 * because the TSN-150 device ID has been released by PLX for us
 */
#define TECNINT_SUBVENDOR_ID  (PLX_VENDOR_ID) // TSN-150 uses a PLX code!!!!
#define TECNINT_TSN150PCI_ID  (0x2277)              // TSN-150 device
#define TECNINT_TSN151PCI_ID  (0x2842)              // TSN-151 device (diagnostic)

/*
 *-----------------------------------------------------------------------
 * TSN150-ISA specific defines
 *-----------------------------------------------------------------------
 */
#define MIX_TSN150_ISA_IO           (0x1F0)  //! TSN150 ISA min IO address
#define MAX_TSN150_ISA_IO           (0x3FF)  //! TSN150 ISA max IO address

/*----------------- ISA IRQ -------------------*/
#define MAX_ISA_IRQ                 (15)   //! Last available IRQ line

/*
 * Register to access to ETN buffer: the REGETN_MEMHIGHPTR is incremented
 * automatically by 1 every time the REGETN_MEMLOWPTR register overflown.
 * The ETN has two 16Kbyte buffers: one buffer is the QUIET buffer and it is
 * available to the PC side for read and write, whilts the other one is the
 * ETN working buffer. The PC sets the ETN records in the QUIET buffer then
 * swaps the WORK and QUIT buffers to enable the new data configuration.
 * The swap it is requested also for read the ETN data caming from slaves:
 * in this way the two buffers must be filled with the same records.
 */
/*
 *-----------------------------------------------------------------------
 * TSN150 IO register map
 *-----------------------------------------------------------------------
 */
#define ETN_MAX_IO_REGS       (8)           // Max ETN IO registers
#define EXT_MAX_IO_REGS       (16)          // Max Extended IO registers

#define ETN_MEM_PTR_LSB       (0x0000)      //___/ Select ETN memory location to
#define ETN_MEM_PTR_MSB       (0x0001)      //   \ be accessed (32K bytes)
```

```
#define ETN_MEM_DATA          (0x0002)       //  ETN memory: read/write data


//-----------------------------------------------------------------------------
//    7654-3210      **** ETN_CTRL_STS ****
//    |||| ||||
//    |||| |||+----<> R/W 1=Enable ETN network
//    |||| ||+-----<> R/W 1=Enable ETN Interrupt
//    |||| |+------<> R/W Select the ETN memory buffer 0 or 1
//    |+++-+--------> R      *** reserved ***
//    +-------------> R    EOT: 1=End of ETN data transmission
//-----------------------------------------------------------------------------
#define ETN_CTRL_STS          (0x0003)       // Controls ETN / Read EOT status

               /*------ ETN_CTRL_STS bit mask ------------*/
#define        ETC_ENA_MSK    (0x01)         // ETN_ENABLE: 1=Enable, 0=Disable
#define        ETC_INT_MSK    (0x02)         // ETN_IRQ   : 1=Enable, 0=Disable
#define        ETC_PHASE_MSK  (0x04)         // ETN_PHASE : 0=Phase0, 1=Phase1
#define        ETC_EOT_MSK    (0x80)         // ETN_EOT   : 1= Tx completed (RD ONLY)


#define ETN_RESET_STATUS      (0x0004)       // Reset IRQ, Start a new ETN cycle


//-----------------------------------------------------------------------------
//    7654-3210        **** ETN_EXT_CTRL ****
//    |||| ||||
//    |||| |+++----<> R/W ETN frequency
//    |||| |            000  12.0    Mbit <-- Or MAX speed (3, 6 or 12Mbit)
//    |||| |            001   6.0    Mbit
//    |||| |            010   3.0    Mbit
//    |||| |            011   1.5    Mbit
//    |||| |            100   750    Kbit
//    |||| |            101   375    Kbit
//    |||| |            110   187    Kbit
//    |||| |            111    93.75 Kbit
//    |||| |
//    |||+-+--------> R      *** reserved ON TSN-150/PC104 and PCI ***
//    |||| +-------->  1= Enable Baud change on OLD ISA board
//    |||+---------->  1= Enable Attribute channel on OLD ISA board
//    |||
//    |++----------<> R/W  ETN refresh mode
//    |                 00   manual refresh: start using ETN_RESET_STATUS
//    |                 01   1seconds automatic refresh    (1)
//    |                 1x   Automatic, continous refresh
//    |
//    +-------------> R      *** reserved ***
//-----------------------------------------------------------------------------
#define ETN_EXT_CTRL          (0x0005)       //  Extended ETN controls

               /*------ ETN_EXT_CTRL bit mask ------------*/
#define  ETE_BAUD_MSK  (0x07)         // -R/W  Bit rate  :

#define        ETE_BAUD_DEF   (0x00) //   000 Default (3.125Mbit OR 6,25 Mbit)
#define        ETE_6M         (0x01) //   001 6,25  Mbit
#define        ETE_3M         (0x02) //   010 3.125 Mbit
#define        ETE_1M5               (0x03) //   011 1.5   Mbit
#define        ETE_750K              (0x04) //   100 750   Kbit
#define        ETE_375K              (0x05) //   101 375   Kbit
#define        ETE_187K              (0x06) //   110 187   Kbit
#define        ETE_93K               (0x07) //   111 93.75 Kbit

#define MAX_ETN_BAUD_CODE     (0x07) // Max ETN BAUD CODE

#define ETE_BAUD_ENA_MSK      (0x08) // -R/W  Enable BAUD change on OLD boards
#define ETE_ATTRIB_ENA_MSK    (0x10) // -R/W  Enable Attribute channel change
#define  ETE_REFRESH_MSK      (0x40) // -R/W  REFRESH: 1=Continuous, 0=One shot

/*
 *----------------------------------------------------------------------
 * TSN150 ETN record defines
 * In one ETN RAM PHASE (16K), we can fil up to 1024 records:
 * the first record #0 is reserved and is used by the ETN
 * controller, whilst the last record MUST be always filled with 0's
 * so the user has rooms for 1022 active recors.
 *----------------------------------------------------------------------
```

```
 */
#define ETN_RAM_SIZE          (32768)           // Size of whole ETN exchange RAM
#define ETN_PHASE_SIZE        (ETN_RAM_SIZE/2)   // Size of a SINGLE exchange page
#define ETN_RECORD_SIZE       (16)              // SIZE OF A SINGLE ETN RECORD
#define ETN_MAX_RECORD        (1024)   // Max number of ETN records.
#define ETN_MAX_USR_RECORD    (1022)   // Max number of User records.
#define ETN_MIN_USR_RECORD    (1)      // User must start from record 1


/*
 * NOTE: The structure MUST BE BYTE-ALLIGNED !!!!
 */
typedef struct _ETN_RECORD
{
       /* ETN Transmission Area */
       UCHAR  etnTxSlaveId;   /* Address of slave: 0=END_OF_RECORD_LIST */
       UCHAR  etxTxSlaveType;  /* Type of slave                         */

       union  {
              DWORD  EtnTxDataL;    /* 32 bit data BIG ENDIAN ORDERED */
              UCHAR  EtnTxDataB[4]; /* Byte data */
       }etnTxData;

       UCHAR  etnTxMode;     /* Transaction mode :
                              *   7654-3210
                              *   |||| |||+----> Transition mode: 0=32 bit, 1=16 bit
                              *   +|+| +++-----> not used: must be 0
                              *    | +---------> RS485 Line selector: 0=Line0, 1=Line1
                              *    +-----------> Media type: 0= Wire, 1=Optical Fiber
                              */

       UCHAR  etnTxReserved; /* Not used */

       /* ETN Reception Area */
       UCHAR  etnRxSlaveId;  /* Address of module received   */
       UCHAR  etxRxSlaveType;/* Type of slave                */

       union  {
              DWORD  EtnRxDataL;    /* 32 bit data BIG ENDIAN ORDERED */
              UCHAR  EtnRxDataB[4]; /* Byte data */
       }etnRxData;

       UCHAR  etnRxMasterId; /* Address of the master received */
       UCHAR  etnRxStatus;   /* Status flags: 0=OK, 0xFF=ERROR */
}ETN_RECORD;

//----------- etnRxStatus code -----------------
#define ETN_STATUS_ERR       (0xFF)  // STATUS: Transition ERROR
#define ETN_STATUS_OK        (0x00)  // STATUS: Transition OK
#define ETN_STATUS_INV       (0x55)  // STATUS: Software marker for transition non done

#define ETN_DATA_HH    (0)     // High Word:High byte data   -- 32bit transition only--
#define ETN_DATA_HL    (1)     // High Word:Low byte  data   -- 32bit transition only--
#define ETN_DATA_LH    (2)     // Low Word: High byte data   -- 16/32bit transition --
#define ETN_DATA_LL    (3)     // Low Word: Low byte data    -- 16/32bit transition --

/*
 *-----------------------------------------------------------------------------
 * WRITE ATTRIBUTE BIT MASK
 *-----------------------------------------------------------------------------
 * 7654-3210
 * |||| |||+------> RecordSize: 0=32bit, 1=16bit
 * |||| +++-------> NU (set to 0)
 * |||+-----------> RS485 Channel: 0= Ch1, 1=Ch2
 * ||+------------> NU (set to 0)
 * |+-------------> Media   : 0= RS485, 1=FIBER
 * +--------------> NU (set to 0)
 *
 *-----------------------------------------------------------------------------
 */
#define ETN_ATTRIB_SIZE16     (0x01)
#define ETN_ATTRIB_SIZE32     (0x00)
#define ETN_ATTRIB_CH0        (0x00)
#define ETN_ATTRIB_CH1        (0x10)
#define ETN_ATTRIB_RS485      (0x00)
#define ETN_ATTRIB_FIBER      (0x40)
```

```
#define ETN_ATTRIB_SIZE_MSK  (0x01)     // Size flag mask
#define ETN_ATTRIB_CH_MSK    (0x10)     // Channel flag mask
#define ETN_ATTRIB_MED_MSK   (0x40)     // Media selected mask
#define ETN_ATTRIB_MASK             (ETN_ATTRIB_SIZE_MSK|ETN_ATTRIB_CH_MSK|ETN_ATTRIB_MED_MSK)

/*
 *----------------------------------------------------------------------
 * TSN150 ETN Slave type
 * The ETN address can be setted from 1 up to 255 (0xFF)
 * The address 0 is reserved and is used for mark the END OF RECORD list
 * The address 255 (0xFF) is also reserved for mark the 'dummy' records
 * used to space regular records of the same slave for timing problems.
 * When a dummy record is used, its type MUST BE 0x00.
 *----------------------------------------------------------------------
 */
#define ETN_MAX_ADR          (255)               // Max number of ETN address

#define ETNADR_NULL          (0x00)              // NULL slave: end of record marker
#define ETNADR_DUMMY         (0xFF)              // Slave address used for DUMMY rec.

#define ETN_TYP_0            (0x00)              //
#define ETN_TYP_1            (0x01)              //
#define ETN_TYP_2            (0x02)              //
#define ETN_TYP_3            (0x03)              //
#define ETN_TYP_4            (0x04)              //
#define ETN_TYP_5            (0x05)              //
#define ETN_TYP_6            (0x06)              //
#define ETN_TYP_7            (0x07)              //
#define ETN_TYP_8            (0x08)              //
#define ETN_TYP_9            (0x09)              //
#define ETN_TYP_A            (0x0A)              //
#define ETN_TYP_B            (0x0B)              //
#define ETN_TYP_C            (0x0C)              //
#define ETN_TYP_D            (0x0D)              //
#define ETN_TYP_E            (0x0E)              //
#define ETN_TYP_F            (0x0F)              //
#define ETN_TYP_NONE  (0xFF)               // NOT AN ETN TYPE

#define ETNTYP_NULL   (ETN_TYP_0)    // Invalid type
#define ETNTYP_MASK   (0x0F)               // Ident mask
#define ETN_16BIT            (ETN_ATTRIB_SIZE16 << 8)
#define ETN_32BIT            (ETN_ATTRIB_SIZE32 << 8)
#define ETN_CHSEL            (ETN_ATTRIB_SIZE32 << 8)

/*
 *----------------------------------------------------------------------
 * TSN150 ETN Slave type
 * The ETN address can be setted from 1 up to 255 (0xFF)
 * The address 0 is reserved and is used for mark the END OF RECORD list
 * The address 255 (0xFF) is also reserved for mark the 'dummy' records
 * used to space regular records of the same slave for timing problems.
 * When a dummy record is used, its type MUST BE 0x00.
 *----------------------------------------------------------------------
 */
#define ETN_ETN16BIT  (ETN_16BIT | ETNTYP_NULL) // RW Generic 16 bit ETN equipment
#define ETN_ETN32BIT  (ETN_32BIT | ETNTYP_NULL) // RW Generic 32 bit ETN equipment

#define ETN_TSR20     (ETN_16BIT | ETN_TYP_5)  // R- TSR20, 16din
#define ETN_TSR31     (ETN_16BIT | ETN_TYP_6)  // -W TSR31, 16dout

#define ETN_TSR40     (ETN_16BIT | ETN_TYP_6)  // RW TSR40, 16din - 16dout
#define ETN_TSR44     (ETN_32BIT | ETN_TYP_6)  // RW TSR44, 32din - 24dout
#define ETN_TSR48     (ETN_32BIT | ETN_TYP_F)  // RW TSR48, 32din - 32dout
#define ETN_TSR51     (ETN_16BIT | ETN_TYP_9)  // RW TSR51, 4 or 8 ain          complex
#define ETN_TSR52     (ETN_16BIT | ETN_TYP_A)  // RW TSR52, 4 ain - 4 Pwm aout  complex
#define ETN_TSR56     (ETN_16BIT | ETN_TYP_8)  // R- TSR56, 8 or 16 ain         complex
#define ETN_TSR57     (ETN_16BIT | ETN_TYP_8)  // R- TSR57, 8 Ain
#define ETN_TSR67     (ETN_32BIT | ETN_TYP_A)  // RW TSR67, 4 Enc.ain, 8din 4aout, 8dout
#define ETN_TSR72     (ETN_32BIT | ETN_TYP_B)  // R- TSR72, 4 Encoder ain      complex
#define ETN_TSR73     (ETN_32BIT | ETN_TYP_B)  // R- TSR73, 4 SSI ain         complex
#define ETN_ETN10     (ETN_16BIT | ETN_TYP_C)  // RW ETN10, 8In, 8out 2 Analog in

       //
       //----------------------------------------------------------------------
       // Clone equipments: different physical format, same working mode
```

```
        //---------------------------------------------------------------------
        //
#define ETN_TSR35       (ETN_TSR31)     // Alias of TSR31
#define ETN_TSR47       (ETN_TSR40)     // Alias of TSR40
#define ETN_TSR447      (ETN_TSR40)     // Alias of TSR40, custom board

#define ETN_ETN16DI     (ETN_TSR20)     // Alias of TSR20, new line
#define ETN_ETN16DO     (ETN_TSR31)    // Alias of TSR31, new line

#define ETN_ETN32       (ETN_TSR40)    // Alias of TSR40, new line
#define ETN_ETN40       (ETN_TSR40)    // Alias of TSR40, new line

#define ETN_TSR51S      (ETN_TSR51)    // Alias of TSR51, small board
#define ETN_ETN51       (ETN_TSR51)    // Alias of TSR51, new line
#define ETN_ETN51S      (ETN_TSR51)    // Alias of TSR51, new line

#define ETN_TSR56S      (ETN_TSR56)    // Alias of TSR56, small board
#define ETN_ETN56       (ETN_TSR56)    // Alias of TSR56, new line
#define ETN_ETN56S      (ETN_TSR56S)   // Alias of TSR56, new line

#define ETN_ETN67       (ETN_TSR67)     // Alias of TSR67, new line, enhanced functions

#define ETN_ETN72       (ETN_TSR72)              // Alias of TSR72, new line
#define ETN_ETN73       (ETN_TSR73)              // Alias of TSR73, new line

                //[------------------------------------------]
                //[ ETN-SPECIFIC TSN150 I/O register access   ]
                //[------------------------------------------]

VOID    tsn150_etnExtSet(IN PUCHAR port, BYTE ext);
BYTE    tsn150_etnExtGet(IN PUCHAR port );
VOID    tsn150_etnBaudSet(IN PUCHAR port, BYTE baud);
BYTE    tsn150_etnBaudGet(IN PUCHAR port);
VOID    tsn150_etnModeSet(IN PUCHAR port, BYTE mode);

VOID    tsn150_etnCtrlSet(IN PUCHAR port, BYTE ctrl);

BYTE    tsn150_etnCtrlGet(IN PUCHAR port);

VOID    tsn150_etnNetSet(IN PUCHAR port, BYTE ena);
BYTE    tsn150_etnNetStat(IN PUCHAR port);

VOID    tsn150_etnIrqSet(IN PUCHAR port, BYTE ena);
BYTE    tsn150_etnIrqStat(IN PUCHAR port);

VOID    tsn150_etnPhaseSet(IN PUCHAR port, BYTE ph);
BYTE    tsn150_etnPhaseGet(IN PUCHAR port);
BYTE    tsn150_etnPhaseSwap(IN PUCHAR port);

BYTE    tsn150_etnEotStat(IN PUCHAR port);
VOID    tsn150_etnStart(IN PUCHAR port);
VOID    tsn150_etnUnlockStart(IN PUCHAR port);



                //[------------------------------------------]
                //[ ETN-SPECIFIC TSN150 PCI MEMORY access     ]
                //[------------------------------------------]

unsigned long  tsn150pci_etnRdl(void* pEtnMem);
void           tsn150pci_etnWrl(void* pEtnMem, unsigned long data);

void* tsn150pci_etnRecPtr(
        void*                pBase,  // ETN PCI memory base
        unsigned short       nRec,   // Number of ETN record
        unsigned short       ofs     // Specific ETN record offset
);
int tsn150pci_etnReadData(
        void*                pBase,          // ETN PCI memory base
        unsigned char  phase,        // ETN memory phase (0 or 1)
        DWORD*               pData,          // Pointer where store data
        unsigned short nRec       // Record to be read (1...MAX)
);

int tsn150pci_etnWriteData(
```

```
        void*                   pBase,          // ETN PCI memory base
        unsigned char  phase,           // ETN memory phase + Access type (bit7=1 16 bit)
        DWORD                   Data,           // data to write
        unsigned short nRec         // Record to be write
        );


ETN_RECORD* tsn150pci_initRecord(
        void*           pBase,  // base of ETN buffer
        unsigned short nRec,    // Record to initialize
        unsigned char  address,// Slave address
        unsigned char  type,    // Slave Ident type
        unsigned char  attrib, // Slave attribute
        unsigned long  ldata    // Tx Initialization data
);

VOID tsn150pci_clearRecord(
                void*   pBase,          // base of ETN buffer (Phase0 or phase1 base)
                unsigned short nRec             // Record to clear
        );

// Gets the first FREE record in ETN RAM.
// First Free can be a DUMMY record or the LAST record of the list (Address=0)
// Return the index of free record
int tsn150pci_etnGetFreeRec(
        IN void*                pBase,  // ETN MEM base
        IN unsigned short       nRec    // ETN start record (0=Default start record)
        );


// Gets the number of active records in ETN RAM
// Return the number of used records, included DUMMYes
int tsn150_etnGetNrec(
    IN void*                    pBase,  // ETN MEM base
    IN unsigned char   phase    // ETN memory phase (0,1)
);

VOID tsn150pci_clearEtnRam(IN void* pBase);

int tsn150_init(IN PUCHAR port, IN void* pBase);

#ifdef __cplusplus
}
#endif

#endif                                          /*  _TSN150IO_H_ */
/**************
* END_OF_FILE *
**************/
```

```
/*!
$Revision: $
//****************************************************************************
//   Software by  Tecnint SrL, VERONA/MERATE(LC) ITALY
//----------------------------------------------------------------------------
// Original Writer: sma
//----------------------------------------------------------------------------
// PRODUCT IDENTIFICATION : TSN150 DRIVER FOR WINDOWS OS
//----------------------------------------------------------------------------
// Module: TSN150IO.CPP
//
//   Interface to TSN150 boards.
//   This file contains all the LOW-LEVEL functions usefull to access the
//   TSN-150 ETN master board, both in ISA or PCI mode.
//   The TPLC-138 is an extension of TSN150 implemented in the TPLC138 board.
//   The TPLC138 register extension is described in the file TPLC138IO.C
//
//
//  TSN-150 PC104(ISA)
//  --------------------
//  The TSN150 ISA board can be accessed througth this set of 8bit registers
//  in the I/O space:
//
//    BASEIO+0x0000  ETN_MEM_PTR_LSB   \___ Select ETN memory location to
//    BASEIO+0x0001  ETN_MEM_PTR_MSB   /    be accessed (32K bytes)
//    BASEIO+0x0002  ETN_MEM_DATA           ETN memory: read/write data
//    BASEIO+0x0003  ETN_CTRL              Controls ETN functions
//    BASEIO+0x0004  ETN_RESET_STATUS       Reset IRQ, Start ETN cycle
//    BASEIO+0x0005  ETN_EXT_CTRL           More ETN controls
//
//  TSN-150 PCI
//  --------------------
//  The TSN150 PCI board has the same functionality of the ISA card in the
//  I/O space but it can also access to ETN memory in the memory space
//  avoiding the I/O pointer methods. The direct memory access is much
//  faster that the I/O access and can improve performance.
//  Another big advantage is the the plug-and-play configuration of the PCI
//  card over the manual configuration of ISA card.
//  The PCI interface chip is the 9050 PLX type:
//           VENDOR ID =  0x10B5
//           DEVICE ID =  0x9050
//
//  TSN150 REGISTERS EXPLANATION
//  ------------------------------
//   ETN_MEM_PTR_LSB/MSB
//     registers used to point the ETN memory location to be accessed
//
//   ETN_MEM_DATA
//     registers used to write or read in the ETN memory location pointed
//     by the ETN_MEM_PTR. Every access to this register will cause an
//     AUTOMATIC INCREMENT of the ETM_MEM_PTR_XXX register(s).
//
//   ETN_CTRL
//    7654-3210
//    |||| |||+----<> R/W 1=Enable ETN network
//    |||| ||+-----<> R/W 1=Enable ETN Interrupt
//    |||| |+------<> R/W Select the ETN memory buffer 0 or 1
//    |+++-++-------> R      *** reserved ***
//    +-------------> R    EOT: 1=End of ETN data transmission
//
//   ETN_RESET_STATUS
//     Any access (Read or Write) to this regis will cause the RESET of
//     the interrupt flag and the START of a NEW ETN CYCLE, if the ETN
//     network is ENABLED.
//
//   ETN_EXT_CTRL  (1)
//    7654-3210
//    |||| |+++----<> R/W ETN frequency  (1)
//    |||| |            000   Default (Max, 6 or 12Mbit)
//    |||| |            001   6      Mbit
//    |||| |            010   3      Mbit
//    |||| |            011   1.5    Mbit
//    |||| |            100   750    Kbit
//    |||| |            101   375    Kbit
//    |||| |            110   187    Kbit
//    |||| |            111    93.75 Kbit
```

```
//    |||| |
//    |||+-+--------> R   TSN150/PC104/ISA: reserved
//    |||| |
//    |||| +-------<> R/W TSN150-ISA: 1=Enable baud rate change
//    |||+---------<> R/W TSN150-ISA: 1=Enable Attribute channel change
//    |||
//    |++----------<> R/W  ETN refresh mode
//    |                 00   manual refresh: start using ETN_RESET_STATUS
//    |                 01   1seconds automatic refresh
//    |                 1x   Automatic refresh, NO INTERRUPT
//    |
//    +-------------> R      *** reserved ***
//
//  (1) Baud rate is configurable on TSN-150/ISA board only enabling the
//      bit 3 !!
//
//  The data are exchanged on the ETN network from a set of record of 16
//  bytes each one stored in the ETN memory.
//  The ETN memory is a 32Kbyte RAM splitted into two buffers, one buffer
//  is ACTIVE whilst the other one is INACTIVE.
//  The INACTIVE buffer is ALWAYS at relative address of 0x0000 up to 0x3FFF
//  and the ACTIVE buffer used by ETN controller is ALWAYS at relative
//  address 0x4000 up to 0x7FFF, also when the buffers are swapped.
//
//  You can store up to 1022 CONSECUTIVE exchange recors in the active/
//  inactive buffer.
//  The FIRST RECORD is RESERVED and is used by the ETN MASTER CONTROLLER

//  The record list MUST BE terminated by a NULL RECORD (16 bytes all set
//  to 0).
//  Structure of ETN record:
//     0x00       Address of ETN Slave (1..127)
//     0x01       Type of SLAVE
//     0x02       Data byte 3 to be trasmit to slave          \
//     0x03       Data byte 2 to be trasmit to slave           \__ 32bit
//     0x04       Data byte 1 to be trasmit to slave \_ 16 bit /
//     0x05       Data byte 0 to be trasmit to slave /        /
//     0x06       Attribute
//     0x07       reserved (must be 0)
//
//     0x08       Returned Slave Address
//     0x09       Returned Type of SLAVE
//     0x0A       Data byte 3 received from slave          \
//     0x0B       Data byte 2 received from slave           \__ 32bit
//     0x0C       Data byte 1 received from slave \_ 16 bit /
//     0x0D       Data byte 0 received from slave /        /
//     0x0E       Returned MASTER station (0)
//     0x0F       Error flag: 0=OK, FF=Error
//
//  The attribute field controls the type or transition:
//   7654-3210
//   |||| |||+-------> Transition: 1=16bit, 0=32bit
//   |||| +++--------> *** reserved ***
//   |||+------------> Channel: 0=CH0, 1=CH1 (RS485 only)
//   ||+-------------> *** reserved ***
//   |+--------------> Media: 0=RS485, 1=Optical fiber
//   +---------------> *** reserved ***
//
// Warning:
//   The ISA/PC-104 TSN140 access is a SLOW 500ns per BYTE, so to
//   increase permormance, some attentions must be payed in writing
//   only the really needed field.
//
//   Use "TECNINT's C Coding Standards" conventions.
//
//  ===> THESE FUNCTIONS ARE NOT PROTECTED AGAINST CONCURRENCY.
//       CONCURRENCY PROTECTION MUST BE ASSURED BY THE CALLER.
//  ===> ON THE TSN150 ISA BOARD THE I/O BASE, THE INTERRUPT LEVEL AND THE
//       ETN SPEED ARE FIXED BY JUMPERS AND CANNOT BE CHANGED BY SOFTWARE.
//
//  ===> ON THE TSN150-PC104(ISA) AND TSN128 BOARD THE I/O BASE, THE INTERRUPT
//       LEVEL ARE FIXED BY JUMPERS AND CANNOT BE CHANGED BY SOFTWARE, BUT THE
//       ETN SPEED IS SOFTWARE-CONFIGURABLE
//
// WARNING:
//
```

```
// Environment info:
//   PC board - ISA or PCI card
//
//   Compiler: MSVC
//   Debugger: MS
//---------------------------------------------------------------------------
// Copyright (c)  Tecnint SrL.
//
// Redistribution and use in source and binary forms, with or without
// modification, are not permitted without a written permission of
// Tecnint SrL.
// 1. Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
// 3. All advertising materials mentioning features or use of this software
//    must display the following acknowledgement:
//    This product includes software developed by the Tecnint SrL
// 4. The name of the Tecnint SrL cannot be used to endorse or promote
//    products derived from this software without specific prior written
//    permission.
//
// THIS SOFTWARE IS PROVIDED BY THE TECNINT SRL  ``AS IS'' AND ANY EXPRESS
// OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
// DISCLAIMED. IN NO EVENT SHALL TECNINT SRL BE LIABLE FOR ANY DIRECT,
// INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
// HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
// STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
// IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//
//***************************************************************************
//                                      VERSION CONTROL HISTORY
//---------------------------------------------------------------------------
$Log: $
//***************************************************************************
*/
#include "wdm1.h"
#include "Ioctl.h"
#include "tsn150.h"

               //[-------------------------------------------]
               //[ LOCAL CONDITIONAL COMPILE SWITCHES        ]
               //[-------------------------------------------]

               //[-------------------------------------------]
               //[ LOCAL DEFINES AND MACROS                  ]
               //[-------------------------------------------]
#ifdef USE_PCI_BYTE_ACCESS
#      define ACC_TYPE       DWORD
#      define REC_LEN        (ETN_RECORD_SIZE/(sizeof(DWORD)))
#      define RAM_LEN        (ETN_RAM_SIZE/ (sizeof(DWORD)))
#else
#      define ACC_TYPE       char
#      define REC_LEN        (ETN_RECORD_SIZE)

#      define RAM_LEN        (ETN_RAM_SIZE)
#endif

               //[-------------------------------------------]
               //[ IMPORTED functions (Externals)            ]
               //[-------------------------------------------]


               //[-------------------------------------------]
               //[ PRIVATE  functions (static locals)        ]
               //[-------------------------------------------]


               //[-------------------------------------------]
               //[ LOCAL variables (static locals)               ]
               //[-------------------------------------------]
```

```
              //[------------------------------------------]
              //[ GLOBAL variables (public )                   ]
              //[------------------------------------------]

              //[------------------------------------------]
              //[ ETN-SPECIFIC TSN150 I/O register access   ]
              //[------------------------------------------]

/*![
//***************************************************************************
// Name: tsn150_etnExtSet                                          Type: PUBLIC utility
//
// Abstract:
//   Set the ETN extended register:
//
//    7654-3210        **** ETN_EXT_CTRL ****
//    |||| |+++----<> R/W ETN frequency
//    |||| |            000   Default (Max,3, 6 or 12Mbit)
//    |||| |            001   6      Mbit
//    |||| |            010   3      Mbit  <---
//    |||| |            011   1.5    Mbit
//    |||| |            100   750    Kbit
//    |||| |            101   375    Kbit
//    |||| |            110   187    Kbit
//    |||| |            111    93.75 Kbit
//    |||| |
//    |||+-+--------> R      *** reserved ***
//    |++----------<> R/W  ETN refresh mode
//    |                 00   manual refresh: start using ETN_RESET_STATUS
//    |                 01   1seconds automatic refresh (ONLY TSN150-ISA)
//    |                 1x   Automatic refresh, NO INTERRUPT
//    |
//    +-------------> R      *** reserved ***
//
// Warning:
//
// Interfaces:
//       port     TSN-150 IO Base Port
//       ext     Extended flags
//
// return: NONE
//***************************************************************************
]*/
VOID tsn150_etnExtSet(IN PUCHAR port, BYTE ext)
{
       out8((port+ETN_EXT_CTRL), ext);
}

/*![
//***************************************************************************
// Name: tsn150_etnExtGet                                          Type: PUBLIC utility
//
// Abstract:
//   Get the ETN extended register.
//
// Warning:
//
// Interfaces:
//       port     TSN-150 IO Base Port
//       ext     Extended flags
//
// return: NONE
//***************************************************************************
]*/
BYTE tsn150_etnExtGet(IN PUCHAR port )
{
       return ( in8((port+ETN_EXT_CTRL)) );
}

/*![
//***************************************************************************
// Name: tsn150_etnBaudSet                                         Type: PUBLIC utility
//
// Abstract:
//   Set the ETN baud rate in the EXTENDED register.
//
```

```
// Warning:
//  ETE_BAUD_ENA_MSK and ETE_ATTRIB_ENA_MSK bits are set for compatibility
//  with OLD ISA board!!!
//
// Interfaces:
//       port     TSN-150 IO Base Port
//       baud    baud rate code: 0 up to 7
//                                 0 - Default (3.125Mbit OR 6,25 Mbit)
//                                 1 - 6,25  Mbit
//                          2 - 3.125 Mbit
//                          3 - 1.5   Mbit
//                          4 - 750   Kbit
//                          5 - 375   Kbit
//                          6 - 187   Kbit
//                          7 - 93.75 Kbit
//
// return: NONE
//****************************************************************************
]*/
VOID tsn150_etnBaudSet(IN PUCHAR port, BYTE baud)
{
       register char rext;

       rext = ( (in8((port+ETN_EXT_CTRL))& ~ETE_BAUD_MSK)|(baud&ETE_BAUD_MSK)
                          | (ETE_BAUD_ENA_MSK | ETE_ATTRIB_ENA_MSK) );

       out8((port+ETN_EXT_CTRL), rext);
}

/*![
//****************************************************************************
// Name: tsn150_etnBaudGet                                         Type: PUBLIC utility
//
// Abstract:
//   Get the ETN baud rate
//
// Warning:
//
// Interfaces:
//       port     TSN-150 IO Base Port
//
// return: baud rate code
//****************************************************************************
]*/
BYTE tsn150_etnBaudGet(IN PUCHAR port)
{
       return( ((in8( (port+ETN_EXT_CTRL))) & ETE_BAUD_MSK) );
}


/*![
//****************************************************************************
// Name: tsn150_etnModeSet                                         Type: PUBLIC utility
//
// Abstract:
//   Set the ETN work mode in the extended register.
//
// Warning:
//
// Interfaces:
//       port     TSN-150 IO Base Port
//       mode     0=Manual refresh 1=Continuous (Automatic)
//
// return: NONE
//****************************************************************************
]*/
VOID tsn150_etnModeSet(IN PUCHAR port, BYTE mode)
{
       register char rext;
       rext = ( (in8((port+ETN_EXT_CTRL))& ~ETE_REFRESH_MSK)|((mode<<6)&ETE_REFRESH_MSK));
       out8((port+ETN_EXT_CTRL), rext);
}

/*![
//****************************************************************************
// Name: tsn150_etnCtrlSet                                         Type: PUBLIC utility
```

```
//
// Abstract:
//   Set the ETN CONTROL register
//
//    7654-3210      **** ETN_CTRL_STS ****
//    |||| ||||
//    |||| |||+----<> R/W 1=Enable ETN network
//    |||| ||+-----<> R/W 1=Enable ETN Interrupt
//    |||| |+------<> R/W Select the ETN memory buffer 0 or 1
//    |+++-+--------> R      *** reserved ***
//    +-------------> R    EOT: 1=End of ETN data transmission
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//      ctrl    Control flags -- see register format ---
//
// return: mode
//***************************************************************************
]*/
VOID tsn150_etnCtrlSet(IN PUCHAR port, BYTE ctrl)
{
       out8((port+ETN_CTRL_STS), ctrl);
}

/*![
//***************************************************************************
// Name: tsn150_etnCtrlGet                                   Type: PUBLIC utility
//
// Abstract:
//   Get the ETN CONTROL register
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: ETN control register flags
//***************************************************************************
]*/
BYTE tsn150_etnCtrlGet(IN PUCHAR port)
{
       return ( in8((port+ETN_CTRL_STS)) );
}


/*![
//***************************************************************************
// Name: tsn150_etnNetSet                                    Type: PUBLIC utility
//
// Abstract:
//   Enable or disable the ETN network.
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//      ena     0=Disable ETN, 1=Enable ETN
//
// return: NONE
//***************************************************************************
]*/
VOID tsn150_etnNetSet(IN PUCHAR port, BYTE ena)
{
       out8((port+ETN_CTRL_STS),
                    (((in8((port+ETN_CTRL_STS))) & ~ETC_ENA_MSK) |      (ena &ETC_ENA_MSK))
              );
}

/*![
//***************************************************************************
// Name: tsn150_etnNetStat                                   Type: PUBLIC utility
//
// Abstract:
//   Get ETN network status
```

```
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: 0=ETN network is disabled, 0 != Network enabled
//***************************************************************************
]*/
BYTE tsn150_etnNetStat(IN PUCHAR port)
{
       return ( ((in8((port+ETN_CTRL_STS))) & ETC_ENA_MSK) );
}


/*![
//***************************************************************************
// Name: tsn150_etnIrqSet                                        Type: PUBLIC utility
//
// Abstract:
//   Enable or disable the ETN interrupt
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//      ena     0=Disable ETN, 1=Enable ETN
//
// return: NONE
//***************************************************************************
]*/
VOID tsn150_etnIrqSet(IN PUCHAR port, BYTE ena)
{
       out8(  (port+ETN_CTRL_STS),
                     (((in8((port+ETN_CTRL_STS))) & ~ETC_INT_MSK) |((ena <<1) & ETC_INT_MSK))
              );
}


/*![
//***************************************************************************
// Name: tsn150_etnIrqStat                                       Type: PUBLIC utility
//
// Abstract:
//   Get ETN network status
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: 0=ETN IRQ is disabled, 0 != ETN IRQ enabled
//***************************************************************************
]*/
BYTE tsn150_etnIrqStat(IN PUCHAR port)
{
       return ( (((in8((port+ETN_CTRL_STS))) & ETC_INT_MSK) >> 1) ) ;
}

/*![
//***************************************************************************
// Name: tsn150_etnPhaseSet                                      Type: PUBLIC utility
//
// Abstract:
//   Set the ACTIVE PHASE buffer.
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//      phase   0=Phase 0, 1=Phase 1
//
// return: NONE
//***************************************************************************
]*/
VOID tsn150_etnPhaseSet(IN PUCHAR port, BYTE ph)
```

```
{
        out8((port+ETN_CTRL_STS),
                        (((in8((port+ETN_CTRL_STS))) & ~ETC_PHASE_MSK) |
                        ((ph <<2) & ETC_PHASE_MSK))
                );
}

/*![
//*****************************************************************************
// Name: tsn150_etnPhaseGet                                       Type: PUBLIC utility
//
// Abstract:
//   Get the ACTIVE PHASE buffer.
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: Phase selected 0 or 1
//*****************************************************************************
]*/
BYTE tsn150_etnPhaseGet(IN PUCHAR port)
{
        return ( (in8((port+ETN_CTRL_STS)) & ETC_PHASE_MSK) >> 2 );
}

/*![
//*****************************************************************************
// Name: tsn150_etnPhaseSwap                                      Type: PUBLIC utility
//
// Abstract:
//   Swap the ACTIVE PHASE buffer.
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: Current active Phase
//*****************************************************************************
]*/
BYTE tsn150_etnPhaseSwap(IN PUCHAR port)
{
        BYTE phase;

        phase = (in8((port+ETN_CTRL_STS)) ^ ETC_PHASE_MSK);  // DEBUG.....

        out8((port+ETN_CTRL_STS), (phase) );

        return ( (in8((port+ETN_CTRL_STS)) & ETC_PHASE_MSK) >> 2 );
}


/*![
//*****************************************************************************
// Name: tsn150_etnEotStat                                        Type: PUBLIC utility
//
// Abstract:
//   Get ETN EOT network status
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: 0= Not EOT, 0 != ETN EOT asserted (all records has been tx)
//*****************************************************************************
]*/
BYTE tsn150_etnEotStat(IN PUCHAR port)
{
        return ( ((in8((port+ETN_CTRL_STS))) & ETC_EOT_MSK) ) ;
}


/*![
```

```
//*****************************************************************************
// Name: tsn150_etnStart                                        Type: PUBLIC utility
//
// Abstract:
//   Start a new ETN transition
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: NONE
//*****************************************************************************
]*/
VOID tsn150_etnStart(IN PUCHAR port)
{
       /*
        * Any Read or Write access is enougth to clear any
        * ETN pending interrupt and to restart a new etn cycle.
        * WE use a WRITE access only to remark that is an active
        * operation.....
        */
       out8((port+ETN_RESET_STATUS), 0x55);
}


/*![
//*****************************************************************************
// Name: tsn150_etnUnlockStart                                          Type: PUBLIC utility
//
// Abstract:
//   Start an ETN cycle executing the unlock procedure first.
//   The ETN150 may occasionally LOCK UP its internal state machine
//   and in such condition, the ETN cycale cannot be restarted again if
//   the TSN150 is not unlocked.
//   To unlock the TSN150 we must disable the ETN NETWORK.
//   Then, we have to enable the Network and start a new cycle.
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//      ena     0=Disable ETN, 1=Enable ETN
//
// return: NONE
//*****************************************************************************
]*/
VOID tsn150_etnUnlockStart(IN PUCHAR port)
{
       // Unlock TSN150 disabling the ETN network
       out8((port+ETN_CTRL_STS),(((in8((port+ETN_CTRL_STS))) & ~ETC_ENA_MSK) ));

       // Enable the ETN network
       out8((port+ETN_CTRL_STS),(((in8((port+ETN_CTRL_STS))) | ETC_ENA_MSK) ));

       // START A NEW ETN CYCLE.....
       out8((port+ETN_RESET_STATUS), 0x55);
}

/*![
//*****************************************************************************
// Name: tsn150_etnMemPtrSet                             Type: PUBLIC utility
//
// Abstract:
//   Set a new ETN memory pointer
//
// Warning:
//
// Interfaces:
//      port    TSN-150 IO Base Port
//      ofs     Offset in ETN memory, from 0 up to ETN_RAM_SIZE
//
// return: OH or ERROR
//*****************************************************************************
]*/
int tsn150_etnMemPtrSet(
```

```
        IN PUCHAR port,
        IN unsigned short ofs
)
{
        if(ofs > ETN_RAM_SIZE)
        {
               return(-1);
        }

        out8((port+ETN_MEM_PTR_LSB), (unsigned char)(ofs) );
        out8((port+ETN_MEM_PTR_MSB), (unsigned char)(ofs >> 8));

        return(0);
}

/*![
//****************************************************************************
// Name: tsn150_etnMemFill                          Type: PUBLIC utility
//
// Abstract:
//   Fill the whole ETN memory
//
// Warning:
//  The user MUST fill memory with 0.
//  Other values are RESERVED to TECNINT for diagnostic purposes.
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: Number of bytes filled OR 0
//****************************************************************************
]*/
int
tsn150_etnMemFill(
        IN PUCHAR port,
        IN unsigned short     ofs,
        IN unsigned char      fill
)
{
        int    j;
        int    n;

        if(0 != (tsn150_etnMemPtrSet(port, ofs)))
        {
               return(0);
        }

        n= (ETN_RAM_SIZE-ofs);

        for(j=0; j <n; j++)
        {
               out8((port+ETN_MEM_DATA), (char)(fill) );
        }
        return( n );
}


/*
```

```
*/
               //[-----------------------------------------]
               //[ ETN-SPECIFIC TSN150 PCI MEMORY access     ]
               //[-----------------------------------------]

/*![
//*************************************************************************
// Name: tsn150pci_etnWrl                                   Type: PUBLIC utility
//
// Abstract:
//   Write a LONG into ETN memory at CURRENT MEMORY POSITION
//
// Warning:
//   ETN memory must be ordered in BIG ENDIAN mode (Motorola)
//
// Interfaces:
//
// return: OK, ERROR
//*************************************************************************
]*/
void tsn150pci_etnWrl(void* pEtnMem, unsigned long data)
{
#ifdef USE_PCI_BYTE_ACCESS
       //
       // Write in TSN150 PCI memory using BYTE access (More slow)
       // Use this way if a LONG access to TSN150 PCI fails or
       // reports some trouble.....
       *((char*)pEtnMem)++ = (unsigned char)(data>>24);
       *((char*)pEtnMem)++ = (unsigned char)(data>>16);
       *((char*)pEtnMem)++ = (unsigned char)(data>>8);
       *((char*)pEtnMem)   = (unsigned char)(data);
#else
       union  {
              DWORD  lEtn;
              BYTE   bEtn[4];
       }etn;

       // In order to minimize the PCI transfert time, a LONG access
       // is preferred....at first, orded data in Big endian mode
       //
       etn.bEtn[ETN_DATA_HH] = (unsigned char)(data>>24);
       etn.bEtn[ETN_DATA_HL] = (unsigned char)(data>>16);
       etn.bEtn[ETN_DATA_LH] = (unsigned char)(data>>8);
       etn.bEtn[ETN_DATA_LL] = (unsigned char)(data);

       *((DWORD*)pEtnMem)=etn.lEtn;  /* LONG WORD PCI ACCESS */
#endif
}


/*![
//*************************************************************************
// Name: tsn150pci_etnRdl                                   Type: PUBLIC utility
//
// Abstract:
//   Read a LONG WORD from ETN memory at CURRENT MEMORY POSITION
//
// Warning:
//   ETN memory is ordered in BIG ENDIAN mode (Motorola)
//
// Interfaces:
//      port    TSN-150 IO Base Port
//
// return: long value
//*************************************************************************
]*/
unsigned long tsn150pci_etnRdl(void* pEtnMem)
{
       union  {
              unsigned long ldata;
              unsigned char bdata[4];

       }data;
#ifdef USE_PCI_BYTE_ACCESS
       //
       // Write in TSN150 PCI memory using BYTE access (More slow)
```

```
        // Use this way if a LONG access to TSN150 PCI fails or
        // reports some trouble.....

        data.bdata[3]=*((char*)pEtnMem)++;
        data.bdata[2]=*((char*)pEtnMem)++;
        data.bdata[1]=*((char*)pEtnMem)++;
        data.bdata[0]=*((char*)pEtnMem);

#else                               /* USE_PCI_BYTE_ACCESS */

        unsigned long ldata;

        ldata = *((DWORD*)pEtnMem);  /* PCI LONG WORD ACCESS */

        data.bdata[0] = (unsigned char)(ldata>>24); /* High byte--> low Byte */
        data.bdata[1] = (unsigned char)(ldata>>16); /* High byte */
        data.bdata[2] = (unsigned char)(ldata>>8);  /* High byte */
        data.bdata[3] = (unsigned char)(ldata);     /* High byte */

#endif                              /* USE_PCI_BYTE_ACCESS */

        return ( data.ldata);
}


/*![
//*****************************************************************************
// Name: tsn150pci_etnRecPtr                                Type: PUBLIC utility
//
// Abstract:
//   Get the address of the specified ETN record
//
// Warning:
//   ETN memory is ordered in BIG ENDIAN mode (Motorola): data in the
//   buffer must be already ordered in BIG ENDIAN MODE!!!
//
// Interfaces:
//
// return: Pointer to ETN record or NULL
//*****************************************************************************
]*/
void*
tsn150pci_etnRecPtr(
        void*                 pBase,  // ETN PCI memory base
        unsigned short        nRec,   // Number of ETN record
        unsigned short        ofs     // Specific ETN record offset
)
{
        if((0 == nRec) || (nRec > ETN_MAX_USR_RECORD)||(NULL==pBase))
        {
                return(NULL);
        }

        pBase =  ((char*)pBase)+ ((nRec*ETN_RECORD_SIZE)+ofs);

        return(pBase);
}


/*![
//*****************************************************************************
// Name: tsn150pci_etnReadData                                      Type: PUBLIC utility
//
// Abstract:
//   Read the 32bit ETN data from a record and clear the transition
//   status byte.
//
// Warning:
//  ETN transition MUST be NOT working beore attemtp to read!!!
//
//  Data is read ONLY if the GOOD TRANSITION code is found in the status.
//  The transition status record is marked with a BAD VALUE before exit.
//
// Interfaces:
```

```
//
// return: ETN transition status (0=OK)
//****************************************************************************
]*/
int tsn150pci_etnReadData(
	void*			pBase,		// ETN PCI memory base
	unsigned char	phase,		// ETN memory phase (0 or 1)
	DWORD*			pData,		// Pointer where store data
	unsigned short nRec		// Record to be read (1...MAX)
)
{
	ETN_RECORD*		pRecord;
	int				status;

	if((NULL== pData) || (NULL==pBase) || ( nRec > ETN_MAX_USR_RECORD) || (0== nRec))
	{
		return(-1);
	}

	pRecord= (ETN_RECORD*)pBase;

	if(0 != phase )
	{
		/*
		 * We have to read a record from ETN phase buffer (buffer1)
		 */
		pRecord += (ETN_PHASE_SIZE/ETN_RECORD_SIZE);
	}

	pRecord += nRec;		// Point to record to be read

	if (0 == ((status=pRecord->etnRxStatus)&0xFF) )
	{
		//
		// This is a VALID ETN transition... read the DATA from ETN
		// memory and return it.
		//
		*pData = tsn150pci_etnRdl((void*)&pRecord->etnRxData.EtnRxDataL);

		//
		// Invalidate the ETN record for the next ETN transition...
		//
		pRecord->etnRxStatus=ETN_STATUS_INV;
	}

	return(status);		// Status of record (0=OK, 0xFF=Fail, 0x55=not done
}

/*![
//****************************************************************************
// Name: tsn150pci_etnWriteData					Type: PUBLIC utility
//
// Abstract:
//   Write the 32bit ETN data INTO an ETN record .
//
// Warning:
//  ETN transition MUST be NOT working beore attemtp to write!!!
//
// Interfaces:
//
// return: ETN transition status (0=OK)
//****************************************************************************
]*/
int tsn150pci_etnWriteData(
	void*			pBase,		// ETN PCI memory base
	unsigned char	phase,		// ETN memory phase + Access type (bit7=1 16 bit)
	DWORD			Data,		// data to write
	unsigned short nRec		// Record to be write
)
{
	ETN_RECORD*		pRecord;
	int				status;

	if((NULL==pBase) || ( nRec > ETN_MAX_USR_RECORD) || (0== nRec))
	{
		return(-1);
```

```
        }

        pRecord= (ETN_RECORD*)pBase;

        if(0 != phase )

        {
                /*
                 * We have to read a record from ETN phase buffer (buffer1)
                 */
                pRecord += (ETN_PHASE_SIZE/ETN_RECORD_SIZE);
        }

        pRecord += nRec;              // Point to record to be read

        tsn150pci_etnWrl((char*)&pRecord->etnRxData.EtnRxDataL, Data );


        return(0);                     // OK
}

*/
/*![
//***************************************************************************
// Name: tsn150_initRecord                                         Type: PUBLIC utility
//
// Abstract:
//   Initialize a new record in the ETN memory
//
// Warning:
//   Parameters are NOT checked. Caller can set any combination of
//   address, type and attributes. Illegal combinations of such params
//   may cause malfunctions.
//
// Interfaces:
//
// return: Pointer of allocated record
//***************************************************************************
]*/
ETN_RECORD*
tsn150pci_initRecord(
        void*                  pBase,         // base of ETN buffer (Phase0 or phase1 base)
        unsigned short nRec,           // Record to initialize
        unsigned char  address,    // Slave address
        unsigned char  type,           // Slave Ident type
        unsigned char  attrib,     // Slave attribute
                               //   7654-3210
                               //   |||| |||+--> Transition mode: 0=32 bit, 1=16 bit
                               //   +|+| +++-----> not used: must be 0
                               //    | +---------> RS485 Line selector: 0=Line0, 1=Line1
                               //    +-----------> Media type: 0= Wire, 1=Optical Fiber
        unsigned long  ldata  // Tx Initialization data
)
{
        ETN_RECORD*            pRecord;
        union  {
                unsigned long ldata;
                unsigned char bdata[4];

        }data;

        if(NULL != (pRecord =(ETN_RECORD*)tsn150pci_etnRecPtr(pBase,nRec,0)))
        {
                pRecord->etnTxSlaveId         = 0;   /* just for safety... */

                /* ETN Reception Area */
                pRecord->etnRxSlaveId         = 0;
                pRecord->etxRxSlaveType       = 0;
                pRecord->etnRxMasterId        = 0;

#ifdef USE_PCI_BYTE_ACCESS
                pRecord->etnRxData.EtnTxDataB[ETN_DATA_HH] = 0;
                pRecord->etnRxData.EtnTxDataB[ETN_DATA_HL] = 0;
                pRecord->etnRxData.EtnTxDataB[ETN_DATA_LH] = 0;
                pRecord->etnRxData.EtnTxDataB[ETN_DATA_LL] = 0;
#else
```

```
		pRecord->etnRxData.EtnRxDataL = 0;
#endif
		pRecord->etnRxStatus          = ETN_STATUS_INV; /* Status flags */

		/* ETN Transmission Area */
		pRecord->etnTxData.EtnTxDataB[ETN_DATA_HH] = (unsigned char)(ldata>>24);
		pRecord->etnTxData.EtnTxDataB[ETN_DATA_HL] = (unsigned char)(ldata>>16);
		pRecord->etnTxData.EtnTxDataB[ETN_DATA_LH] = (unsigned char)(ldata>>8);
		pRecord->etnTxData.EtnTxDataB[ETN_DATA_LL] = (unsigned char)(ldata);

		pRecord->etnTxMode      = attrib;
		pRecord->etnTxReserved  = 0;

		pRecord->etxTxSlaveType =type;       /* Type of slave
		  */
		pRecord->etnTxSlaveId   =address; /* Address of slave: ENDLIST */
	}
	return(pRecord);
}

/*![
//****************************************************************************
// Name: tsn150_clearRecord                                        Type: PUBLIC utility
//
// Abstract:
//   Clear a specified record
//
// Warning:
//   Parameters are NOT checked. Caller can set any combination of
//   address, type and attributes. Illegal combinations of such params
//   may cause malfunctions.
//
// Interfaces:
//
// return: Pointer of allocated record
//****************************************************************************
]*/
VOID
tsn150pci_clearRecord(
	void*               pBase,        // base of ETN buffer (Phase0 or phase1 base)
	unsigned short nRec          // Record to clear
)
{
	ACC_TYPE*      pRecord;
	int            j;

	if(NULL != (pRecord =(ACC_TYPE*)tsn150pci_etnRecPtr(pBase,nRec,0)))
	{
		/*
		 * Clear 16 bytes (record length)
		 */
		for(j=0; j < REC_LEN ; j++)
		{
			*pRecord++ =0;
		}
	}
}

/*![
//****************************************************************************
// Name: tsn150pci_clearEtnRam                                        Type: PUBLIC utility
//
// Abstract:
//   Clear the whole ETN RAM area
//
// Warning:
//
// Interfaces:
//
// return: NONE
//****************************************************************************
]*/
VOID tsn150pci_clearEtnRam(void* pBase)
{
	ACC_TYPE*      pRam;
	int            j;
```

```
        pRam=(ACC_TYPE*)pBase;

        for(j=0; j < (RAM_LEN) ; j++)
        {
                *pRam++ = 0;
        }
}

*/
/*![
//*************************************************************************
// Name: tsn150_etnGetFree                          Type: PUBLIC utility
//
// Abstract:
//   Get a free record position into the ETN memory.
//   A free position is a record that has as Address the 0 OR a DUMMY
//   record (0xFF marker)
//
// Warning:
//
// Interfaces:
//
// return: Free record index or 0 if no one is free.
//*************************************************************************
]*/
int
tsn150pci_etnGetFreeRec(
    IN void*                  pBase,  // ETN MEM base
        IN unsigned short     nRec    // ETN start record (0=Default start record)
)
{
        ETN_RECORD*           pRecord;
        int                   j;
        unsigned char  addr;

        if(0==nRec)
        {
                nRec++;       // START FROM RECORD 1 !!!!!
        }

        if(NULL != (pRecord =(ETN_RECORD*)tsn150pci_etnRecPtr(pBase,nRec,0)))
        {
                for(j=nRec; j < ETN_MAX_USR_RECORD; j++)
                {
                        addr= pRecord->etnTxSlaveId;

                        if((0==addr) || (ETNADR_DUMMY==addr))
                        {
                                //
                                // OK! a FREE record has been FOUND
                                //
                                return(j);                    // OK! Free record found....
                        }

                        pRecord++;     /* Point to Next record */
                }
        }
        return(0);            /* NO FREE RECORDS */
}
```

```
/*![
//***************************************************************************
// Name: tsn150_etnGetNrec                          Type: PUBLIC utility
//
// Abstract:
//   Get the number of used records
//
// Warning:
//
// Interfaces:
//
// return: Number of active records.
//***************************************************************************
]*/
int
tsn150_etnGetNrec(
    IN void*                  pBase,  // ETN MEM base
        IN unsigned char      phase   // ETN memory phase (0,1)
)
{
        ETN_RECORD*           pRecord;
        int                   j;
        unsigned short ofs;

        ofs=0;
        if(0 != phase)
        {
                ofs=ETN_PHASE_SIZE;
        }
        ofs += (1*ETN_RECORD_SIZE);           // Start from RECORD 1

        pRecord= (ETN_RECORD*)( ((char*)pBase)+ofs);

        for(j=0; j < ETN_MAX_USR_RECORD; j++)
        {
                if(0== pRecord->etnTxSlaveId)
                {
                        //
                        // OK! LAST record has been FOUND
                        //
                        break;
                }
                pRecord++;     /* Point to Next record */
        }
        return(j);
}

/*![
//***************************************************************************
// Name: tsn150_init                        Type: PUBLIC utility
//
// Abstract:
//   Initialize a TSN150 board accessing its I/O registers (ISA and PCI)
//
// Warning:
//
// Interfaces:
//
// return: OK or a failure code
//***************************************************************************
]*/
int tsn150_init(IN PUCHAR port, IN void* pBase)
{
        tsn150_etnCtrlSet(port,0);           // Disable control
        tsn150_etnExtSet(port, 0);           // Disable EXTENDED register

        tsn150_etnBaudSet(port,ETE_1M5); // Set default baud rate 1.5Mbit

        //
        // Now read back the the baud code to be sure that this is a
        // valid I/O register.
        // This is not a completly safe check,but ....
        //
        if(ETE_1M5 != tsn150_etnBaudGet(port))
        {
                /*
```

```
                 * BAUD REGISTER READ BACK FAILURE!!!!!
                 */
                if(0 != tsn150_etnCtrlGet(port))
                {
                        return(-1);             // Hardware failure
                }
        }

        /*
         * Clear the WHOLE ETN RAM
         */
        if(NULL != pBase)
        {
                // Use the PCI memory access to clear RAM !!
                tsn150pci_clearEtnRam(pBase);
        }
        else
        {
                // Use the ISA I/O access to clear RAM !!
                tsn150_etnMemFill(port,0,0);
        }
        return(0);
}
/***************
 * END_OF_FILE *
 **************/
```

8.4.3. Esempio configurazione TSN150/PCI come TSN150/ISA in DOS

In DOS, Windows95 o Windows98 e' possibile configurare la scheda TSN150/PCI in modo da emulare completamente una scheda TSN150/ISA.
Per fare questo è necessario modificare la porta di I/O e la linea di interrupt automaticamente assegnate alla scheda TSN150/PCI dal BIOS del PC.
In particolare la porta di accesso deve essere modificata per ricadere nel range di validià delle porte ISA (da 0x10 a 0x3FF), mentre la line di interrupt può essere lasciata al valore assegnato.

Questa operazione deve essere effettuata tenendo conto delle porte di I/O e degli interrupt effettivamente disponibili sul proprio sistema: **si ricorda che questa operazione, se mal condotta, potrebbe causare mafunzionamenti di periferiche ISA o instabilità del sistema.**

A seguto sono riportati i sorgenti di una utilità che permette di configurare la porta di I/O e la linea di interrupt di una scheda TSN150/PCI in modo tale da poterla successivamente utilizzare come una normale TSN150ISA.

Programmi sviluppati per DOS o Windows95/98 possono così essere utilizzati senza modifiche anche montando una scheda TSN150PCI al posto di una scheda ISA.

```
/*
$Revision: $
//***************************************************************************
//   Software by  Tecnint SrL, VERONA/MERATE(LC) ITALY
//--------------------------------------------------------------------------
// Original Writer: sma
//--------------------------------------------------------------------------
// PRODUCT IDENTIFICATION :  TSN150PCI
//--------------------------------------------------------------------------
// Module: PCILIB.H
//
//   Definitions for TSN150 PCI Library
//
// Warning:
//
//--------------------------------------------------------------------------
// Copyright (c)  Tecnint SrL.
//
// Redistribution and use in source and binary forms, with or without
// modification, are not permitted without a written permission of
// Tecnint SrL.
// 1. Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
// 3. All advertising materials mentioning features or use of this software
//    must display the following acknowledgement:
//    This product includes software developed by the Tecnint SrL
// 4. The name of the Tecnint SrL cannot be used to endorse or promote
//    products derived from this software without specific prior written
//    permission.

//
// THIS SOFTWARE IS PROVIDED BY THE TECNINT SRL ``AS IS'' AND ANY EXPRESS
// OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
// DISCLAIMED. IN NO EVENT SHALL TECNINT SRL BE LIABLE FOR ANY DIRECT,
// INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
// HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
// STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
// IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//
//**************************************************************************
//                                       VERSION CONTROL HISTORY
//--------------------------------------------------------------------------
$Log: $
//***************************************************************************
*/

/*
 ***************************************************************************
 *  General Constants                                                      *
 ***************************************************************************
 */
#define TRUE    1
#define FALSE   0

typedef unsigned char byte;     /* 8-bit  */
typedef unsigned short word;    /* 16-bit */
typedef unsigned long dword;    /* 32-bit */

#define CARRY_FLAG 0x01         /* 80x86 Flags Register Carry Flag bit */

/*
 ***************************************************************************
 *   PCI Functions                                                         *
 ***************************************************************************
 */
#define PCI_FUNCTION_ID           0xb1
#define PCI_BIOS_PRESENT          0x01
#define FIND_PCI_DEVICE           0x02
#define FIND_PCI_CLASS_CODE       0x03
```

```
#define GENERATE_SPECIAL_CYCLE    0x06
#define READ_CONFIG_BYTE          0x08
#define READ_CONFIG_WORD          0x09
#define READ_CONFIG_DWORD         0x0a
#define WRITE_CONFIG_BYTE         0x0b
#define WRITE_CONFIG_WORD         0x0c
#define WRITE_CONFIG_DWORD        0x0d

/*
 ***************************************************************************
 *   PCI Return Code List                                                  *
 ***************************************************************************
 */
#define SUCCESSFUL               0x00
#define NOT_SUCCESSFUL           0x01
#define FUNC_NOT_SUPPORTED       0x81
#define BAD_VENDOR_ID            0x83
#define DEVICE_NOT_FOUND         0x86
#define BAD_REGISTER_NUMBER      0x87

/*
 ***************************************************************************
 *   PCI Configuration Space Registers                                     *
 ***************************************************************************
 */                                    // EEPROM PLX-DEF VALUE
#define PCI_CS_VENDOR_ID         (0x00)     //  0      9050
#define PCI_CS_DEVICE_ID         (0x02) //  2      10B5
#define PCI_CS_COMMAND           (0x04) //
#define PCI_CS_STATUS            (0x06) //
#define PCI_CS_REVISION_ID       (0x08)     //     4      000x (class code: rev. ID NOT downlodable)
#define PCI_CS_CLASS_CODE        (0x0A) //  6      0680 (class-code)
#define PCI_CS_CACHE_LINE_SIZE   (0x0c)     //
#define PCI_CS_MASTER_LATENCY    (0x0d) //
#define PCI_CS_HEADER_TYPE       (0x0e) //
#define PCI_CS_BIST              (0x0f) //
#define PCI_CS_BASE_ADDRESS_0    (0x10) //
#define PCI_CS_BASE_ADDRESS_1    (0x14) //
#define PCI_CS_BASE_ADDRESS_2    (0x18)     //          ISA_MEMCS_BASE
#define PCI_CS_BASE_ADDRESS_3    (0x1c)     //          ISA_IOCS_BASE
#define PCI_CS_BASE_ADDRESS_4    (0x20)     //          ISA_CSRAM_BASE
#define PCI_CS_BASE_ADDRESS_5    (0x24)     //          ISA_CSROM_BASE
#define PCI_CS_SUBVENDOR_ID           (0x2C) //  A      10B5
#define PCI_CS_SUBSYSTEM_ID      (0x2E)     //  8      9050-1
#define PCI_CS_EXPANSION_ROM     (0x30) //
#define PCI_CS_INTERRUPT_LINE    (0x3c)     //  E      Int. Pin
#define PCI_CS_INTERRUPT_PIN     (0x3d) //
#define PCI_CS_MIN_GNT           (0x3e) //
#define PCI_CS_MAX_LAT           (0x3f) //

#define PCI_CTRL_REG                  (0x50) // PLX EEPROM / control register

/*
 ***************************************************************************
 *  Specific PLX PCI9050 Operation Register Offsets                        *
 ***************************************************************************
 */
#define PCR_ISA_MEMCS_BASE           (0x18) // Local address space 0
#define PCR_ISA_IOCS_BASE            (0x1C) // Local address space 1
#define PCR_ISA_CSRAM_BASE           (0x20) // Local address space 2
#define PCR_ISA_CSROM_BASE           (0x24) // Local address space 3
#define PCR_INT_LINE                 (0x3C) // Interrupt line
#define PCR_CTRL_REG                 (0x50) // EEPROM / control register

#define EE_CLK_BIT                          (24L)
#define EE_ENA_BIT                          (25L)
#define EE_DO_BIT                           (26L)
#define EE_DI_BIT                           (27L)

#define EE_ENA_MASK                         (1L <<EE_ENA_BIT)
#define EE_CLK_MASK                         (1L <<EE_CLK_BIT)
#define EE_DO_MASK                          (1L <<EE_DO_BIT)
#define EE_DI_MASK                          (1L <<EE_DI_BIT)

#define PLX_VENDOR_ID                (0x10B5)       // PLX vendor id
#define PLX_DEVICE_ID                (0x9050)       // PLX 9050 Chip Id
```

```
#define PLX_SUBVENDOR_ID             (0x10B5)       // PLX sub-vendor id
#define PLX_SUBSYSTEM_ID             (0x9050)       // PLX subsystem Id (0x9050-1)

#define TECNINT_SUBID_TSN150  (0x2277)       // PLX assigned subdevice code (TSN150)

#define TECNINT_SUBVENDOR_ID  (PLX_SUBVENDOR_ID)
#define TECNINT_TSN150PCI_ID  (TECNINT_SUBID_TSN150)

       /* PLX EEPROM OFFSET */
#define PLX_EE_SUBVENDOR_ID           (0x0A)
#define PLX_EE_SUBSYSTEM_ID      (0x08)

/*
 ***************************************************************************
 *  Specific AMCC Operation Register Offsets                               *
 ***************************************************************************
 */
#define AMCC_OP_REG_OMB1         0x00
#define AMCC_OP_REG_OMB2         0x04
#define AMCC_OP_REG_OMB3         0x08
#define AMCC_OP_REG_OMB4         0x0c
#define AMCC_OP_REG_IMB1         0x11
#define AMCC_OP_REG_IMB2         0x14
#define AMCC_OP_REG_IMB3         0x18
#define AMCC_OP_REG_IMB4         0x1c
#define AMCC_OP_REG_FIFO         0x20
#define AMCC_OP_REG_MWAR         0x24
#define AMCC_OP_REG_MWTC         0x28
#define AMCC_OP_REG_MRAR         0x2c
#define AMCC_OP_REG_MRTC         0x30
#define AMCC_OP_REG_MBEF         0x34
#define AMCC_OP_REG_INTCSR       0x38
#define AMCC_OP_REG_MCSR         0x3c
#define AMCC_OP_REG_MCSR_NVDATA  (AMCC_OP_REG_MCSR + 2) /* Data in byte 2 */
#define AMCC_OP_REG_MCSR_NVCMD   (AMCC_OP_REG_MCSR + 3) /* Command in byte 3 */

/*
 ****************************************************************************
 *   AMCCLIB Prototypes                                                     *
 ****************************************************************************
 */
int pci_bios_present(byte *hardware_mechanism,
                              word *interface_level_version,
                              byte *last_pci_bus_number);

int find_pci_device(word device_id,
                       word vendor_id,
                       word index,
                       byte *bus_number,
                       byte *device_and_function);

int find_pci_class_code(dword class_code,
                       word index,
                       byte *bus_number,
                       byte *device_and_function);

int generate_special_cycle(byte bus_number,
                             dword special_cycle_data);

int read_configuration_byte(byte bus_number,
                              byte device_and_function,
                              byte register_number,
                              byte *byte_read);

int read_configuration_word(byte bus_number,
                              byte device_and_function,
                              byte register_number,
                              word *word_read);

int read_configuration_dword(byte bus_number,
                               byte device_and_function,
                               byte register_number,
                               dword *dword_read);

int write_configuration_byte(byte bus_number,
                               byte device_and_function,
```

```
                                byte register_number,
                                byte byte_to_write);

int write_configuration_word(byte bus_number,
                                byte device_and_function,
                                byte register_number,
                                word word_to_write);

int write_configuration_dword(byte bus_number,
                                     byte device_and_function,
                                     byte register_number,
                                     dword dword_to_write);

void outpd(word port, dword value);

dword inpd(word port);

void insb(word port, void *buf, int count);
void insw(word port, void *buf, int count);
void insd(word port, void *buf, int count);

void outsb(word port, void *buf, int count);
void outsw(word port, void *buf, int count);
void outsd(word port, void *buf, int count);

/**************
* END_OF_FILE *
**************/
```

```
/*
$Revision: $
//*************************************************************************
//   Software by  Tecnint SrL, VERONA/MERATE(LC) ITALY
//------------------------------------------------------------------------
// Original Writer: Bar
//------------------------------------------------------------------------
// PRODUCT IDENTIFICATION :  TSN150/PCI
//------------------------------------------------------------------------
// Module: MAIN.c
//
// Configura scheda TSN150/PCI in modalita' compatibile con TSN150/ISA
//
// Warning:
//
//------------------------------------------------------------------------
// Copyright (c)  Tecnint SrL.
//
// Redistribution and use in source and binary forms, with or without
// modification, are not permitted without a written permission of
// Tecnint SrL.
// 1. Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
// 3. All advertising materials mentioning features or use of this software
//    must display the following acknowledgement:
//    This product includes software developed by the Tecnint SrL
// 4. The name of the Tecnint SrL cannot be used to endorse or promote
//    products derived from this software without specific prior written
//    permission.
//
// THIS SOFTWARE IS PROVIDED BY THE TECNINT SRL  ``AS IS'' AND ANY EXPRESS
// OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
// DISCLAIMED. IN NO EVENT SHALL TECNINT SRL BE LIABLE FOR ANY DIRECT,
// INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
// HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
// STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
// IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//
//************************************************************************
//                                       VERSION CONTROL HISTORY
//------------------------------------------------------------------------
$Log: $
//*************************************************************************
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "pcilib.h"

/*---------------------------------------------------
 Area dati locali
---------------------------------------------------*/
const char VERSION_INFO[] = "TECNINT HTE srl - TSN150/PCI utility, R1.0,"__DATE__", "__TIME__  ;

#define NUM_OF_PCI_DEV        (4)                   // Max PCI devices
typedef struct pxidev
{
       unsigned long  reg0;     // Configuration register
       unsigned long   base;     // I/O BASE
       unsigned int   irqline;  // Interrupt line
       unsigned char  nBus;     // Bus number
       unsigned char  devId;    // Device Id
}TSN150PCI;
static TSN150PCI      lv_pxidev[NUM_OF_PCI_DEV];
#define ESC           (0x1b)
static byte    lv_curId=0;
```

```
/*
//***************************************************************************
// Name:  setTSN150PCIbase                                              Type: LOCAL
//
// Abstract:
//   Update the PCI I/O BASE address
//
// Warning:
//
// Interfaces:
// return:
//***************************************************************************
*/
void setTSN150PCIbase(int devid, dword base)
{
	if(devid<  NUM_OF_PCI_DEV)
	{
		(void)write_configuration_dword(
				lv_pxidev[devid].nBus,
				lv_pxidev[devid].devId,      // device_and_function,

				PCR_ISA_IOCS_BASE,           // register_number,
				(dword)base
		);
		lv_pxidev[devid].base= base;        // device_and_function,
	}
}

/*
//***************************************************************************
// Name:  setTSN150PCIirq                               Type: LOCAL
//
// Abstract:
//   Update the PCI IRQ level
//
// Warning:
//
// Interfaces:
// return:
//***************************************************************************
*/
void setTSN150PCIirq(int devid, unsigned int irq)
{
	if(devid<  NUM_OF_PCI_DEV)
	{
		(void)write_configuration_byte(
				lv_pxidev[devid].nBus,
				lv_pxidev[devid].devId,      // device_and_function,
				PCR_INT_LINE,                // register_number,
				(byte)irq
			);
		lv_pxidev[devid].irqline = irq;
	}
}

/*
//***************************************************************************
// Name:  pciScanPlx                         Type: utility
//
// Abstract:
//   Scan the PCI bus in order to detect a PLX PCI9050 device, that is
//   used for the TSN150/PCI board.
//   Uses the PCI-BIOS settings to configure the TSN150/PCI board.
//
// Warning:
//   Attach to LAST TSN150/PCI board found.
//
// Interfaces:
//
// return: OK/ERROR
//***************************************************************************
*/
int pciScanPlx(
	int isaBase,          /* ISA IO BASE                                         */
	int isaGap,                  /* ISA IO BASE offset for next board */
	int isaIrq                   /* ISA IRQ                                             */
```

```
)
{
        int     result, found,j;
        byte    busn  = 0;
        byte    devfun= 0;
        dword   reg   = 0;

        byte    hwmet = 0;
        byte    last  = 0;
        word    ifl   = 0;
        byte   temp  = 0;
        byte   first = 0;

       for(j=0; j < NUM_OF_PCI_DEV; j++)
       {
              lv_pxidev[j].reg0    = 0;
              lv_pxidev[j].base    = 0;
              lv_pxidev[j].irqline = 0;
              lv_pxidev[j].devId   = 0;    // Device Id
              lv_pxidev[j].nBus    = 0;    // Device Id
       }

       if(isaBase > 0x3FF)
       {
              printf("\nWARNING: 0x%0X It is Not a valid ISA IO PORT\n\n", isaBase);
       }

       if(isaBase)
       {
              if(isaGap & 0x1)
                 isaGap++;

              if(isaGap> 0x100)
              {
                     isaGap=0x100;
              }
              else if(0==isaGap)
              {
                     isaGap=0x10;          /* Offset for next board mapping */
              }
       }

       result= pci_bios_present(
                              (byte*)&hwmet,    // byte *hardware_mechanism,
                              (word*)&ifl,      //interface_level_version,

                              (byte*)&last);    // last_pci_bus_number

        if(SUCCESSFUL==result)
        {
              printf("\n-------------------------------------------------------------------------");
              printf("\nPCI bios found, hw_mech:%X, Interf. Lev.  Ver: %x.%x, last PCI bus: %x\n",
                              hwmet, ((ifl>>8)&0xFF), ((ifl)&0xFF), last );

              for(j=0, found=0; j< NUM_OF_PCI_DEV; j++)
              {
                     result= find_pci_device(
                              PLX_DEVICE_ID,       // device_id,
                              PLX_VENDOR_ID,              // vendor_id,
                              j,                   // dev. index
                             (byte*)&busn,       // PCI Bus number
                             (byte*)&temp       // PCI Device Id
                            );

                     if(SUCCESSFUL==result)
                     {
                            lv_curId=j;

                            lv_pxidev[j].devId   = temp;    // Device Id
                            lv_pxidev[j].nBus    = busn;    // Bus number

                            devfun = temp;

                            found++;
                            printf("\nFind PXI[%u] of %u, bus num:%X, dev_and_fun: %x",
                                           (j+1), found, busn,devfun);
```

```
                              // REG0-1(W)= Device ID
                              // REG2-3(W)= Vendor ID
                              result=read_configuration_dword(
                                                        busn,                // bus_number,
                                                        devfun,              // device_and_function,
                                                        PCI_CS_VENDOR_ID, // register_number,
                                                        (dword*)&reg         //
                                      );
                              lv_pxidev[j].reg0 = reg;

                              printf("\nDEVICE&VENDOR: %lX, ", reg);

                              // REG8-9(W)= Class Code (revision)
                              // REGA-B(W)= Class Code (revision)
                              result=read_configuration_dword(
                                                        busn,        // bus_number,
                                                        devfun,      // device_and_function,
                                                        PCI_CS_REVISION_ID,   // register_number,
                                                        (dword*)&reg                //
                                      );
                              printf("CLASS:%lX, ", reg);

                              // REG2C-2D(W)= Subsystem vendor ID --> TECNINT code
                              // REG2E-2F(W)= Subsystem ID        --> TSN250  code
                              result=read_configuration_dword(
                                                        busn,   // bus_number,
                                                        devfun,    // device_and_function,
                                                        PCI_CS_SUBVENDOR_ID,  // register_number,
                                                        (dword*)&reg                //
                                      );

                              printf("SUB_ID:%lX", reg);

                              if ( (TECNINT_SUBVENDOR_ID != (reg & 0xFFFF)) ||
                                        (TECNINT_TSN150PCI_ID != ((reg>>16) & 0xFFFF))
                                 )
                              {
                                      printf("\n==> SUBsys & Vendor ID are not TECNINT codes...");
                                      printf("\n==> Please Set these value in EEEROM: ADDR[%u]= %X,
ADDR[%u]= %X ",
                                                                      PLX_EE_SUBVENDOR_ID,
TECNINT_SUBVENDOR_ID,PLX_EE_SUBSYSTEM_ID, TECNINT_TSN150PCI_ID );
                              }

                              //
                              // Read PCI device I/O BASE
                              //
                              if (SUCCESSFUL== (result=read_configuration_dword(
                                                busn,                              // bus_number,
                                                devfun,                    // device_and_function,
                                                PCR_ISA_IOCS_BASE,    // register_number,
                                                (dword*)&reg )
                                        ))
                              {
                                      printf("\nPLX ISA_IOCS_BASE = 0x%08lX", (reg &0xFFFFFFFCL) );

                                      reg &= 0xFFFC;         // mask bit0,bit1 (PCI space)

                                      if(isaBase)
                                      {
                                              if(reg != isaBase)
                                              {
                                                      reg=isaBase;   /* remap board in ISA space */
                                                      setTSN150PCIbase(j, reg);
                                                      printf("...changed PCI IO BASE to ISA BASE:
0x%02X", isaBase );
                                              }

                                              if(isaBase > 0x3FF)
                                              {
                                                      printf("\n===> WARNING: 0x%0X, It is not a Legacy
ISA IO PORT!!!\n\n", isaBase);
                                              }
                                              isaBase += isaGap;
                                      }
```

```
                              lv_pxidev[j].base = reg;
                      }

                      //
                      // Read PCI device Interrupt line
                      //
                      if (SUCCESSFUL== read_configuration_byte(
                                                                  busn,
              // bus_number,
                                                                  devfun,
        // device_and_function,
                                                                  PCR_INT_LINE,
                                                                  (byte*)&reg )
                                              )
                      {
                              reg &= 0xFF;

                              printf("\nPLX INTERRUPT_LINE= %02X", ((byte)reg) );

                              if(isaIrq)
                              {
                                      if((0==first) && (reg != isaIrq))
                                      {
                                              reg=isaIrq;
                                              setTSN150PCIirq(j, isaIrq);
                                              printf("...Changed IRQ to : %02X", isaIrq );
                                              first = 1;
                                      }
                                      else
                                      {
                                              printf("...Not changed");
                                      }
                              }
                              lv_pxidev[j].irqline = (unsigned int)reg;
                      }
              }
        }
        if(0==found)
              printf("\nWARNING! TSN150/PCI chip not found. ");
    }
    else
    {
        printf("\nWARNING! PC-BIOS not detected:");
        printf("\n==> The machine BIOS is too old or the program is running under an usupported
OS.");
    }
    printf("\n---------------------------------------------------------------------------");

    return (found);
}

/*
//***************************************************************************
// Name:  main                                                    Type: PROGRAM ENTRY
//
// Abstract:
//
// Warning:
//
// Interfaces:
//
// return:
//***************************************************************************
*/
int main(int argc, char **argv)
{
    int         i;
    int         nBoard;
    int     isaBase=0;          /* ISA IO BASE                               */
    int     isaGap=0x10;    /* ISA IO BASE offset for next board */
    int     isaIrq=0;           /* ISA IRQ                                   */
    char    *pArg0;
    char    c;

    /************************************************/
```

```
	printf("\n%s\n", &VERSION_INFO[0]);


	pArg0 = argv[0];	/* Program name and path */

	if (argc> 1)
	{
		for (i=1;i<argc;i++)
		{
			const char *pLine= argv[i];

			if ( (*pLine) =='-')
			{
				pLine++;
				while( (*pLine)== ' ') pLine++;	/* Skip blanks */
				c=*pLine;
				pLine++;
				while( (*pLine)== ' ') pLine++;	/* Skip blanks */

				switch(c)
				{
					case 'p':	/* SET IO PORT BASE */
					{
						isaBase = (int)strtoul(pLine, NULL, 0);
						printf("\nREQUESTED I/O BASE PORT: 0x%0X", isaBase);
					}break;

					case 'i':	/* SET IRQ */
					{
						isaIrq = (int)strtoul(pLine, NULL, 0);

						if((isaIrq > 15) || (isaIrq < 5))
						{
							printf("\nINVALID IRQ LINE ......: 0x%0X", isaIrq);
							isaIrq=0;	/* Do not change default */
						}
						else
							printf("\nREQUESTED IRQ LINE ....: 0x%0X", isaIrq);
					}break;

					case 'g':	/* SET PORT GRANULARITY */
					{
						isaGap = (int)strtoul(pLine, NULL, 0);
						if(isaGap& 0x1)
							isaGap++;		/* Odd number are not allowed
*/
						printf("\nREQUESTED PORT GAP ....: %u", isaGap);
					}break;

					default:
					{
						printf("\n%c, invalid option", c);
					}
					case '?':
					{
						printf("\n%s", &VERSION_INFO[0]);
						printf("\nUsage:");
						printf("\n%s -p[port] -i[irq] -g[port_gap]\n", pArg0);
						printf("\nWhere:");
						printf("\n  -p	Assign the ISA legacy port, ie 0x300");
						printf("\n  -i	Assign the ISA legacy IRQ , ie 9");
						printf("\n  -g	Assign the ISA legacy port gap for
multiple TSN150PCI board(16)");
						printf("\n\nExample: ");
						printf("\n%s -p0x310 -i10 -g16\n", pArg0);
					}break;
				}
			}

		}//end_for
	}
	else
	{
		printf("\nTSN150/PCI DOS Diagnostic: ");
	}
```

```
        if(0 != (nBoard=pciScanPlx(isaBase,isaGap, isaIrq)))
        {
                printf("\n\nFound %u TSN150/PCI device(s)\n", nBoard);
        }
        else
        {
                //
                // PCI bios not found OR PCI device NOt found.
                // Let have operator a choice to manually configure the TSN150 board!
                //
                printf("\nTSN150 PCI device not found.\n");
        }


        return(0);
}

/****
EOF
*****/
```

```
/*
$Revision: $
//***************************************************************************
//   Software by  Tecnint SrL, VERONA/MERATE(LC) ITALY
//--------------------------------------------------------------------------
// Original Writer: Sma / AMCLIB.C
//--------------------------------------------------------------------------
// PRODUCT IDENTIFICATION :  TSN150/PCI
//--------------------------------------------------------------------------
// Module: PCILIB.C
//
//      Define a C interface to the PCI BIOS, derived from AMCLIB.C
//
//      Functions Defined:
//
//         PCI_BIOS_PRESENT
//         FIND_PCI_DEVICE
//         FIND_PCI_CLASS_CODE
//         READ_CONFIGURATION_BYTE
//         READ_CONFIGURATION_WORD
//         READ_CONFIGURATION_DWORD
//         WRITE_CONFIGURATION_BYTE
//         WRITE_CONFIGURATION_WORD
//         WRITE_CONFIGURATION_DWORD
//         OUTPD
//         INPD
//         INSB
//         INSW
//         INSD
//         OUTSB
//         OUTSW
//         OUTSD
//
//      Local Functions
//
//         READ_CONFIGURATION_AREA

//      WRITE_CONFIGURATION_AREA
//
//  References:
//
//      [1] "PCI BIOS Specification", Rev. 2.1, PCI Special Interest Group.
//
//      [2] "Pentium Pro Family Developer's Manual -
//           Volume 2: Programmer's Reference Manual",
//               Intel Corporation, Order Number 000901-001,
//        or similar recent x86      instruction set reference.
//
//
// Warning:
//   Porting for Microsoft compiler was not completed.
//   This module can be used ONLY with BORLAND C compilers.
//
//--------------------------------------------------------------------------
// Copyright (c)  Tecnint SrL.
//
// Redistribution and use in source and binary forms, with or without
// modification, are not permitted without a written permission of
// Tecnint SrL.
// 1. Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
// 3. All advertising materials mentioning features or use of this software
//    must display the following acknowledgement:
//    This product includes software developed by the Tecnint SrL
// 4. The name of the Tecnint SrL cannot be used to endorse or promote
//    products derived from this software without specific prior written
//    permission.
//
// THIS SOFTWARE IS PROVIDED BY THE TECNINT SRL  ``AS IS'' AND ANY EXPRESS
// OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
// DISCLAIMED. IN NO EVENT SHALL TECNINT SRL BE LIABLE FOR ANY DIRECT,
// INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
```

```
// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
// HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
// STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
// IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//
//**************************************************************************
//                                      VERSION CONTROL HISTORY
//--------------------------------------------------------------------------
$Log: $
//***************************************************************************
*/

/****************************************************************************/
/*                                                                          */
/* Include Files                                                            */
/*                                                                          */
/****************************************************************************/
#include <dos.h>
#include <stddef.h>
#include "pcilib.h"

//---------- PLX PCI9050 Configuration registers --------------
#define PCR_ISA_MEMCS_BASE          (0x18) // Local address space 0
#define PCR_ISA_IOCS_BASE           (0x1C) // Local address space 1
#define PCR_ISA_CSRAM_BASE          (0x20) // Local address space 2
#define PCR_ISA_CSROM_BASE          (0x24) // Local address space 3
#define PCR_INT_LINE                       (0x3C) // Interrupt line

#define PLX_VENDOR_ID               (0x10B5)      // PLX vedor id
#define PLX_DEVICE_ID               (0x9050)      // PLX 9050 Chip Id


/****************************************************************************/
/*                                                                          */
/* Local Prototypes                                                         */
/*                                                                          */
/****************************************************************************/

static
int read_configuration_area(
              byte          function,
              byte          bus_number,
              byte          device_and_function,
              byte          register_number,
              dword*  data
);

static
int write_configuration_area(
              byte   function,
              byte   bus_number,
              byte   device_and_function,
              byte   register_number,
              dword  value
);

/****************************************************************************/
/*                                                                          */
/* Define macros to obtain individual bytes from a word register            */
/*                                                                          */
/****************************************************************************/

#define HIGH_BYTE(rax) (rax >> 8)
#define LOW_BYTE(rax) (rax & 0xff)


/****************************************************************************/
/*                                                                          */
/*  PCI_BIOS_PRESENT                                                        */
/*                                                                          */
/* Purpose: Determine the presence of the PCI BIOS                          */
/*                                                                          */
/* Inputs: None                                                             */
/*                                                                          */
```

```
/* Outputs:                                                                  */
/*                                                                           */
/*    byte *hardware_mechanism                                               */
/*       Identifies the hardware characteristics used by the platform.      */
/*       Value not assigned if NULL pointer.                                 */
/*          Bit 0 - Mechanism #1 supported                                   */
/*          Bit 1 - Mechanism #2 supported                                   */
/*                                                                           */
/*    word *interface_level_version                                          */
/*      PCI BIOS Version - Value not assigned if NULL pointer.               */
/*              High Byte - Major version number                             */
/*              Low Byte  - Minor version number                             */
/*                                                                           */
/*      byte *last_pci_bus_number                                            */
/*          Number of last PCI bus in the system. Value not assigned if NULL*/
/*       pointer                                                             */
/*                                                                           */
/*    Return Value - Indicates presence of PCI BIOS                          */
/*       SUCCESSFUL - PCI BIOS Present                                       */
/*       NOT_SUCCESSFUL - PCI BIOS Not Present                               */
/*                                                                           */
/****************************************************************************/
int pci_bios_present(
       byte*  hardware_mechanism,
       word*  interface_level_version,
       byte*  last_pci_bus_number
)

{
       int            ret_status;         /* Function Return Status. */
       byte           bios_present_status;/* Indicates if PCI bios present */
       dword  pci_signature;     /* PCI Signature ('P', 'C', 'I', ' ') */

       word           rax, rbx, rcx, flags;  /* Temporary variables to hold register values */

       /* Load entry registers for PCI BIOS */

#ifndef _MSC_VER              /* Not microsoft compiler */
       /* BORLAND */
       _AH = PCI_FUNCTION_ID;
       _AL = PCI_BIOS_PRESENT;

       /* Call PCI BIOS Int 1Ah interface */
       geninterrupt(0x1a);

       /* Save registers before overwritten by compiler usage of registers */
       rax = _AX;
       rbx = _BX;
       rcx = _CX;

       pci_signature = _EDX;
       flags = _FLAGS;

#else
__asm {
       mov            AH,PCI_FUNCTION_ID    ;
       mov            AL,PCI_BIOS_PRESENT   ;
       int            0x1A                          ; PCI bios call

       /* Save registers before overwritten by compiler usage of registers */
       mov            rax, AX;
       mov            rbx, BX;
       mov            rcx, CX;
       mov            pci_signature,EDX;
       PUSHF;
       POP            ax
       mov            flags,AX;
}
#endif

       bios_present_status = HIGH_BYTE(rax);

       /* First check if CARRY FLAG Set, if so, BIOS not present */
       if ((flags & CARRY_FLAG) == 0) {

              /* Next, must check that AH (BIOS Present Status) == 0 */
```

```
                if (bios_present_status == 0) {

         /* Check bytes in pci_signature for PCI Signature */
         if ((pci_signature & 0xff)         == 'P' &&
                  ((pci_signature >> 8) & 0xff)  == 'C' &&
                  ((pci_signature >> 16) & 0xff) == 'I' &&
                  ((pci_signature >> 24) & 0xff) == ' ') {

                 /* Indicate to caller that PCI bios present */
                 ret_status = SUCCESSFUL;

                 /* Extract calling parameters from saved registers */
                 if (hardware_mechanism != NULL) {
                       *hardware_mechanism = LOW_BYTE(rax);
                 }
                 if (interface_level_version != NULL) {
                       *interface_level_version = rbx;
                 }
                 if (last_pci_bus_number != NULL) {
                       *last_pci_bus_number = LOW_BYTE(rcx);
                 }
         }
                }
                else {
         ret_status = NOT_SUCCESSFUL;
                }
        }
        else {
                ret_status = NOT_SUCCESSFUL;
        }

        return (ret_status);
}

/****************************************************************************/
/*                                                                          */
/*  FIND_PCI_DEVICE                                                         */
/*                                                                          */
/* Purpose: Determine the location of PCI devices given a specific Vendor   */
/*          Device ID and Index number.  To find the first device, specify  */
/*          0 for index, 1 in index finds the second device, etc.           */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    word device_id                                                        */
/*       Device ID of PCI device desired                                    */
/*                                                                          */
/*    word vendor_id                                                        */
/*       Vendor ID of PCI device desired                                    */
/*                                                                          */
/*    word index                                                            */
/*       Device number to find (0 - (N-1))                                  */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*    byte *bus_number                                                      */
/*       PCI Bus in which this device is found                              */
/*                                                                          */
/*    byte *device_and_function                                             */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits     */
/*                                                                          */
/*    Return Value - Indicates presence of device                           */
/*       SUCCESSFUL - Device found                                          */
/*       NOT_SUCCESSFUL - BIOS error                                        */
/*       DEVICE_NOT_FOUND - Device not found                                */
/*       BAD_VENDOR_ID - Illegal Vendor ID (0xffff)                         */
/*                                                                          */
/****************************************************************************/
int find_pci_device(
        word   device_id,
        word   vendor_id,
        word   index,
        byte*  bus_number,
        byte*  device_and_function
)
{
```

```
        int             ret_status;     /* Function Return Status */

        word            rax, rbx, flags; /* Temporary variables to hold register values */

        /* Load entry registers for PCI BIOS */

#ifndef _MSC_VER             /* Not microsoft compiler */
        _CX = device_id;
        _DX = vendor_id;
        _SI = index;
        _AH = PCI_FUNCTION_ID;
        _AL = FIND_PCI_DEVICE;

        /* Call PCI BIOS Int 1Ah interface */
        geninterrupt(0x1a);

        /* Save registers before overwritten by compiler usage of registers */
        rax = _AX;
        rbx = _BX;
        flags = _FLAGS;
#else
__asm {
        mov             CX,device_id;
        mov             DX,vendor_id;
        mov             SI,index      ;

        mov             AH,PCI_FUNCTION_ID    ;
        mov             AL,FIND_PCI_DEVICE    ;
        int             0x1A                          ; PCI bios call
        ; /* Save registers before overwritten by compiler usage of registers */
        mov             rax,AX
        mov             rbx,BX
        PUSHF
        POP             ax
        mov             flags,AX
        }
#endif
        /* First check if CARRY FLAG Set, if so, error has occurred */
        if ((flags & CARRY_FLAG) == 0) {

               /* Get Return code from BIOS */
               ret_status = HIGH_BYTE(rax);
               if (ret_status == SUCCESSFUL) {

         /* Assign Bus Number, Device & Function if successful */
         if (bus_number != NULL) {
                *bus_number = HIGH_BYTE(rbx);
         }
         if (device_and_function != NULL) {
                *device_and_function = LOW_BYTE(rbx);
         }
               }
        }
        else {
          ret_status = NOT_SUCCESSFUL;
        }

        return (ret_status);
}


/****************************************************************************/
/*                                                                          */
/*  FIND_PCI_CLASS_CODE                                                     */
/*                                                                          */
/* Purpose: Returns the location of PCI devices that have a specific Class  */
/*          Code.                                                           */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    word class_code                                                       */
/*       Class Code of PCI device desired                                   */
/*                                                                          */
/*    word index                                                            */
/*       Device number to find (0 - (N-1))                                  */
/*                                                                          */
```

```
/* Outputs:                                                            */
/*                                                                     */
/*    byte *bus_number                                                 */
/*       PCI Bus in which this device is found                         */
/*                                                                     */
/*    byte *device_and_function                                        */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits     */
/*                                                                     */
/*    Return Value - Indicates presence of device                      */
/*       SUCCESSFUL - Device found                                     */
/*       NOT_SUCCESSFUL - BIOS error                                   */
/*       DEVICE_NOT_FOUND - Device not found                           */
/*                                                                     */
/***********************************************************************/
int find_pci_class_code(
               dword  class_code,
               word   index,
               byte*  bus_number,
               byte*  device_and_function
)
{
       int ret_status;     /* Function Return Status */
       word rax, rbx, flags; /* Temporary variables to hold register values */

#ifndef _MSC_VER             /* Not microsoft compiler */
       /* Load entry registers for PCI BIOS */
       _ECX = class_code;
       _SI = index;
       _AH = PCI_FUNCTION_ID;
       _AL = FIND_PCI_CLASS_CODE;

       /* Call PCI BIOS Int 1Ah interface */
       geninterrupt(0x1a);

       /* Save registers before overwritten by compiler usage of registers */
       rax = _AX;
       rbx = _BX;
       flags = _FLAGS;
#else
       __asm {
               mov            CX,index;
               mov            SI, CX ;
               mov            CX,class_code;


               mov            AH,PCI_FUNCTION_ID           ;
               mov            AL,FIND_PCI_CLASS_CODE;

               int            0x1A                                 ; PCI bios call

               mov            rax,AX;
               mov            rbx,BX;
               pushf                 ;
               pop            ax
               mov            flags,ax;
       }

#endif

       /* First check if CARRY FLAG Set, if so, error has occurred */
       if ((flags & CARRY_FLAG) == 0) {

               /* Get Return code from BIOS */
               ret_status = HIGH_BYTE(rax);
               if (ret_status == SUCCESSFUL) {

               /* Assign Bus Number, Device & Function if successful */
                       if (bus_number != NULL) {
                             *bus_number = HIGH_BYTE(rbx);
                       }
                       if (device_and_function != NULL) {
                             *device_and_function = LOW_BYTE(rbx);
                      }
               }
       }
       else {
```

```
                ret_status = NOT_SUCCESSFUL;
        }

        return (ret_status);
}

/****************************************************************************/
/*                                                                          */
/*  READ_CONFIGURATION_BYTE                                                 */
/*                                                                          */
/* Purpose: Reads a byte from the configuration space of a specific device  */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    byte bus_number                                                       */
/*       PCI bus to read configuration data from                            */
/*                                                                          */
/*    byte device_and_function                                              */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits     */
/*                                                                          */
/*    byte register_number                                                  */
/*       Register Number of configuration space to read                     */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*    byte *byte_read                                                       */
/*       Byte read from Configuration Space                                 */
/*                                                                          */
/*    Return Value - Indicates presence of device                           */
/*       SUCCESSFUL - Device found                                          */
/*       NOT_SUCCESSFUL - BIOS error                                        */
/*       BAD_REGISTER_NUMBER - Invalid Register Number                      */
/*                                                                          */
/****************************************************************************/
int read_configuration_byte(
                byte    bus_number,
                byte    device_and_function,
                byte    register_number,
                byte*   byte_read
)
{
        int ret_status; /* Function Return Status */
        dword data;

        /* Call read_configuration_area function with byte data */
        ret_status = read_configuration_area(READ_CONFIG_BYTE,
                                        bus_number,
                                        device_and_function,
                                        register_number,
                                        &data);
        if (ret_status == SUCCESSFUL) {
                /* Extract byte */
                *byte_read = (byte)(data & 0xff);
        }

        return (ret_status);
}


/****************************************************************************/
/*                                                                          */
/*  READ_CONFIGURATION_WORD                                                 */
/*                                                                          */
/* Purpose: Reads a word from the configuration space of a specific device  */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    byte bus_number                                                       */
/*       PCI bus to read configuration data from                            */
/*                                                                          */
/*    byte device_and_function                                              */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits     */
/*                                                                          */
/*    byte register_number                                                  */
/*       Register Number of configuration space to read                     */
/*                                                                          */
```

```
/* Outputs:                                                            */
/*                                                                     */
/*    word *word_read                                                  */
/*       Word read from Configuration Space                            */
/*                                                                     */
/*    Return Value - Indicates presence of device                      */
/*       SUCCESSFUL - Device found                                     */
/*       NOT_SUCCESSFUL - BIOS error                                   */
/*       BAD_REGISTER_NUMBER - Invalid Register Number                 */
/*                                                                     */
/***********************************************************************/
int read_configuration_word(
               byte   bus_number,
               byte   device_and_function,
               byte   register_number,
               word*  word_read
)
{
       int ret_status; /* Function Return Status */
       dword data;

       /* Call read_configuration_area function with word data */
       ret_status = read_configuration_area(READ_CONFIG_WORD,
                                     bus_number,
                                     device_and_function,
                                     register_number,
                                     &data);
       if (ret_status == SUCCESSFUL) {

               /* Extract word */
               *word_read = (word)(data & 0xffff);
       }

       return (ret_status);
}

/***********************************************************************/
/*                                                                     */
/*  READ_CONFIGURATION_DWORD                                           */
/*                                                                     */
/* Purpose: Reads a dword from the configuration space of a specific device */
/*                                                                     */
/* Inputs:                                                             */
/*                                                                     */
/*    byte bus_number                                                  */
/*       PCI bus to read configuration data from                       */
/*                                                                     */
/*    byte device_and_function                                         */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits */
/*                                                                     */
/*    byte register_number                                             */
/*       Register Number of configuration space to read                */
/*                                                                     */
/* Outputs:                                                            */
/*                                                                     */
/*    dword *dword_read                                                */
/*       Dword read from Configuration Space                           */
/*                                                                     */
/*    Return Value - Indicates presence of device                      */
/*       SUCCESSFUL - Device found                                     */
/*       NOT_SUCCESSFUL - BIOS error                                   */
/*       BAD_REGISTER_NUMBER - Invalid Register Number                 */
/*                                                                     */
/***********************************************************************/
int read_configuration_dword(

       byte           bus_number,
       byte           device_and_function,
       byte           register_number,
       dword*  dword_read
)
{
       int ret_status; /* Function Return Status */
       dword data;

       /* Call read_configuration_area function with dword data */
```

```
        ret_status = read_configuration_area(READ_CONFIG_DWORD,
                                             bus_number,
                                             device_and_function,
                                             register_number,
                                             &data);
        if (ret_status == SUCCESSFUL) {

               /* Extract dword */
               *dword_read = data;
        }

        return (ret_status);
}

/****************************************************************************/
/*                                                                          */
/*  READ_CONFIGURATION_AREA                                                 */
/*                                                                          */
/* Purpose: Reads a byte/word/dword from the configuration space of a       */
/*          specific device                                                 */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    byte bus_number                                                       */
/*       PCI bus to read configuration data from                            */
/*                                                                          */
/*    byte device_and_function                                              */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits     */
/*                                                                          */
/*    byte register_number                                                  */
/*       Register Number of configuration space to read                     */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*    dword *dword_read                                                     */
/*       Dword read from Configuration Space                                */
/*                                                                          */
/*    Return Value - Indicates presence of device                           */
/*       SUCCESSFUL - Device found                                          */
/*       NOT_SUCCESSFUL - BIOS error                                        */
/*       BAD_REGISTER_NUMBER - Invalid Register Number                      */
/*                                                                          */
/****************************************************************************/
static int read_configuration_area(
               byte          function,
               byte          bus_number,
               byte          device_and_function,
               byte          register_number,
               dword*  data
)
{
        int ret_status; /* Function Return Status */
        word rax, flags; /* Temporary variables to hold register values */
        dword ercx;      /* Temporary variable to hold ECX register value */

#ifndef _MSC_VER              /* Not microsoft compiler */

        /* Load entry registers for PCI BIOS */
        _BH = bus_number;
        _BL = device_and_function;
        _DI = register_number;
   _AH = PCI_FUNCTION_ID;
   _AL = function;

   /* Call PCI BIOS Int 1Ah interface */
   geninterrupt(0x1a);

   /* Save registers before overwritten by compiler usage of registers */
   ercx = _ECX;
   rax = _AX;
   flags = _FLAGS;
#else
__asm {
    mov        BH,bus_number;
    mov        BL,device_and_function;
        mov           DI,register_number;
```

```
    mov        AH, PCI_FUNCTION_ID   ;
        mov           AL,function;
        int           0x1A                              ; PCI bios call

        mov           ercx,ECX;
        mov           rax,AX;
        pushf                ;
        pop           ax
        mov           flags,ax;
        }
#endif
   /* First check if CARRY FLAG Set, if so, error has occurred */
   if ((flags & CARRY_FLAG) == 0) {

      /* Get Return code from BIOS */
      ret_status = HIGH_BYTE(rax);

      /* If successful, return data */
               if (ret_status == SUCCESSFUL) {
         *data = ercx;
      }
        }
   else {
      ret_status = NOT_SUCCESSFUL;
   }


   return (ret_status);
}

/****************************************************************************/
/*                                                                          */
/*  WRITE_CONFIGURATION_BYTE                                                */
/*                                                                          */
/* Purpose: Writes a byte to the configuration space of a specific device   */

/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    byte bus_number                                                       */
/*       PCI bus to write configuration data to                             */
/*                                                                          */
/*    byte device_and_function                                              */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits     */
/*                                                                          */
/*    byte register_number                                                  */
/*       Register Number of configuration space to write                    */
/*                                                                          */
/*    byte byte_to_write                                                    */
/*       Byte to write to Configuration Space                               */
/*                                                                          */
/* Outputs:                                                                 */

/*                                                                          */
/*    Return Value - Indicates presence of device                           */
/*       SUCCESSFUL - Device found                                          */
/*       NOT_SUCCESSFUL - BIOS error                                        */
/*       BAD_REGISTER_NUMBER - Invalid Register Number                      */
/*                                                                          */
/****************************************************************************/
int write_configuration_byte(
               byte bus_number,
               byte device_and_function,
               byte register_number,
               byte byte_to_write
)
{
        int ret_status; /* Function Return Status */

        /* Call write_configuration_area function with byte data */
        ret_status = write_configuration_area(WRITE_CONFIG_BYTE,
                                   bus_number,
                                   device_and_function,
                                   register_number,
                                   byte_to_write);
        return (ret_status);
```

```
}

/**************************************************************************/
/*                                                                        */
/*  WRITE_CONFIGURATION_WORD                                              */
/*                                                                        */
/* Purpose: Writes a word to the configuration space of a specific device   */
/*                                                                        */
/* Inputs:                                                                */
/*                                                                        */
/*    byte bus_number                                                     */
/*       PCI bus to read configuration data from                          */
/*                                                                        */
/*    byte device_and_function                                            */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits   */
/*                                                                        */
/*    byte register_number                                                */
/*       Register Number of configuration space to read                   */
/*                                                                        */
/*    word word_to_write                                                  */
/*       Word to write to Configuration Space                             */
/*                                                                        */
/* Outputs:                                                               */
/*                                                                        */
/*    Return Value - Indicates presence of device                         */
/*       SUCCESSFUL - Device found                                        */
/*       NOT_SUCCESSFUL - BIOS error                                      */
/*       BAD_REGISTER_NUMBER - Invalid Register Number                    */
/*                                                                        */
/**************************************************************************/
int write_configuration_word(
               byte bus_number,
               byte device_and_function,
               byte register_number,
               word word_to_write
)
{
       int ret_status; /* Function Return Status */

       /* Call read_configuration_area function with word data */
       ret_status = write_configuration_area(WRITE_CONFIG_WORD,
                                    bus_number,
                                    device_and_function,
                                    register_number,
                                    word_to_write);
       return (ret_status);
}

/**************************************************************************/
/*                                                                        */
/*  WRITE_CONFIGURATION_DWORD                                             */
/*                                                                        */
/* Purpose: Reads a dword from the configuration space of a specific device */
/*                                                                        */
/* Inputs:                                                                */
/*                                                                        */
/*    byte bus_number                                                     */
/*       PCI bus to read configuration data from                          */
/*                                                                        */
/*    byte device_and_function                                            */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits   */
/*                                                                        */
/*    byte register_number                                                */
/*       Register Number of configuration space to read                   */
/*                                                                        */
/*    dword dword_to_write                                                */
/*       Dword to write to Configuration Space                            */
/*                                                                        */
/* Outputs:                                                               */
/*                                                                        */
/*    Return Value - Indicates presence of device                         */
/*       SUCCESSFUL - Device found                                        */
/*       NOT_SUCCESSFUL - BIOS error                                      */
/*       BAD_REGISTER_NUMBER - Invalid Register Number                    */
/*                                                                        */
```

```
/**************************************************************************/
int write_configuration_dword(
       byte bus_number,
       byte device_and_function,
       byte register_number,
       dword dword_to_write
)
{
       int ret_status; /* Function Return Status */

       /* Call write_configuration_area function with dword data */
       ret_status = write_configuration_area(WRITE_CONFIG_DWORD,
                                      bus_number,
                                      device_and_function,
                                      register_number,
                                      dword_to_write);
       return (ret_status);
}

/**************************************************************************/
/*                                                                        */
/*  WRITE_CONFIGURATION_AREA                                              */
/*                                                                        */
/* Purpose: Writes a byte/word/dword to the configuration space of a      */
/*          specific device                                               */
/*                                                                        */
/* Inputs:                                                                */
/*                                                                        */
/*    byte bus_number                                                     */
/*       PCI bus to read configuration data from                          */
/*                                                                        */
/*    byte device_and_function                                            */
/*       Device Number in upper 5 bits, Function Number in lower 3 bits   */
/*                                                                        */
/*    byte register_number                                                */
/*       Register Number of configuration space to read                   */
/*                                                                        */
/*    dword value                                                         */
/*       Value to write to Configuration Space                            */
/*                                                                        */
/* Outputs:                                                               */
/*                                                                        */
/*    Return Value - Indicates presence of device                         */
/*       SUCCESSFUL - Device found                                        */
/*       NOT_SUCCESSFUL - BIOS error                                      */
/*       BAD_REGISTER_NUMBER - Invalid Register Number                    */
/*                                                                        */
/**************************************************************************/
static int write_configuration_area(
              byte   function,
              byte   bus_number,
              byte   device_and_function,
              byte   register_number,
              dword value
)
{
       int ret_status; /* Function Return Status */
       word rax, flags; /* Temporary variables to hold register values */

#ifndef _MSC_VER             /* Not microsoft compiler */

       /* Load entry registers for PCI BIOS */
       _BH = bus_number;
       _BL = device_and_function;
       _ECX = value;
       _DI = register_number;
       _AH = PCI_FUNCTION_ID;
       _AL = function;

       /* Call PCI BIOS Int 1Ah interface */
       geninterrupt(0x1a);

       /* Save registers before overwritten by compiler usage of registers */
       rax = _AX;
       flags = _FLAGS;
#else
```

```
__asm {
       mov           BH,bus_number;
        mov          BL,device_and_function;
        mov          ECX,value;
        mov          DI,register_number;
       mov           AH, PCI_FUNCTION_ID;
       mov           AL,function;
       int           0x1A                                ; PCI bios call
       mov           rax,AX;
       pushf;
       pop           ax;

       mov           flags,ax;

       }

#endif
       /* First check if CARRY FLAG Set, if so, error has occurred */
       if ((flags & CARRY_FLAG) == 0) {

              /* Get Return code from BIOS */
              ret_status = HIGH_BYTE(rax);
       }
       else {
              ret_status = NOT_SUCCESSFUL;
       }

       return (ret_status);
}

/****************************************************************************/
/*                                                                          */
/*  OUTPD                                                                   */
/*                                                                          */
/* Purpose: Outputs a DWORD to a hardware port                              */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    word port                                                             */
/*       hardware port to write DWORD to                                    */
/*                                                                          */
/*    dword value                                                           */
/*       value to be written to port                                        */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*    None                                                                  */
/*                                                                          */
/****************************************************************************/
void outpd(word port, dword value)
{
#ifndef _MSC_VER             /* Not microsoft compiler */

       _DX  = port;
       _EAX = value;

       // 0x66 below is the operand-size override prefix for x86 processors.
       // Since the 0x66 instruction prefix *complements* the D bit in the
       // segment descriptor, this code assumes that we are running in a 16-bit
       // (DOS?) segment.  It must be modified to work in a 32-bit segment!

       /* Since Borland asm cannot generate OUT  DX, EAX must force in */

   __emit__(0x66, 0xEF);
#else
__asm {
       mov           DX,port
       mov           EAX,value
       out           DX,EAX
       }
#endif
}

/****************************************************************************/
/*                                                                          */
/*  INPD                                                                    */
```

```
/*                                                                         */
/* Purpose: Inputs a DWORD from a hardware port                           */
/*                                                                         */
/* Inputs:                                                                 */
/*                                                                         */
/*    word port                                                            */
/*       hardware port to write DWORD to                                   */
/*                                                                         */
/* Outputs:                                                                */
/*                                                                         */
/*    dword value                                                          */
/*       value read from the port                                          */
/*                                                                         */
/***************************************************************************/
dword inpd(word port)

{
#ifndef _MSC_VER             /* Not microsoft compiler */

   /* Set DX register to port number to be input from */
   _DX = port;

   // 0x66 below is the operand-size override prefix for x86 processors.
   // Since the 0x66 instruction prefix *complements* the D bit in the
   // segment descriptor, this code assumes that we are running in a 16-bit
   // (DOS?) segment.  It must be modified to work in a 32-bit segment!

   /* Since Borland asm cannot generate IN  EAX, DX, must force in */
       __emit__(0x66, 0xED);

   return(_EAX);
#else
__asm {
       mov            DX,port
       in             EAX,DX
       }
#endif

}

/***************************************************************************/
/*                                                                         */
/*  INSB                                                                   */
/*                                                                         */
/* Purpose: Inputs a string of BYTEs from a hardware port                 */
/*                                                                         */
/* Inputs:                                                                 */
/*                                                                         */
/*    word port                                                            */
/*       hardware port to read from                                        */
/*                                                                         */
/*    void *buf                                                            */
/*       Buffer to read data into                                          */
/*                                                                         */
/*    int count                                                            */
/*       Number of BYTEs to read                                           */
/*                                                                         */
/* Outputs:                                                                */
/*                                                                         */
/*    None                                                                 */
/*                                                                         */
/***************************************************************************/
void insb(word port, void *buf, int count)
{
#ifndef _MSC_VER             /* Not microsoft compiler */
   _ES = FP_SEG(buf);   /* Segment of buf */
   _DI = FP_OFF(buf);   /* Offset of buf  */
   _CX = count;         /* Number to read */
   _DX = port;          /* Port           */
   asm   REP INSB;
#else
__asm {
       mov            DI,OFFSET buf;
       mov            AX, SEGMENT buf;
       push   AX
       pop            ES;
```

```
       mov            CX,count;
       REP            INSB;
       }
#endif
}

/***************************************************************************/
/*                                                                         */
/*  INSW                                                                   */
/*                                                                         */
/* Purpose: Inputs a string of WORDs from a hardware port                  */
/*                                                                         */
/* Inputs:                                                                 */
/*                                                                         */
/*    word port                                                            */
/*       hardware port to read from                                        */
/*                                                                         */
/*    void *buf                                                            */
/*       Buffer to read data into                                          */
/*                                                                         */
/*    int count                                                            */
/*       Number of WORDs to read                                           */
/*                                                                         */
/* Outputs:                                                                */
/*                                                                         */
/*    None                                                                 */
/*                                                                         */
/***************************************************************************/
void insw(word port, void *buf, int count)
{
#ifndef _MSC_VER             /* Not microsoft compiler */
   _ES = FP_SEG(buf);   /* Segment of buf */
   _DI = FP_OFF(buf);   /* Offset of buf  */
   _CX = count;         /* Number to read */
   _DX = port;          /* Port           */
   asm   REP INSW;
#else

__asm {
       mov            DI,OFFSET buf ;   /* Offset of buf  */
       mov            ES,SEGMENT buf;   /* Segment of buf */
       mov            CX,count;         /* Number to read */
       mov            DX,port;          /* Port           */
       REP            INSW;
       }

#endif
}

/***************************************************************************/
/*                                                                         */
/*  INSB                                                                   */
/*                                                                         */
/* Purpose: Inputs a string of DWORDs from a hardware port                 */
/*                                                                         */
/* Inputs:                                                                 */
/*                                                                         */
/*    word port                                                            */
/*       hardware port to read from                                        */
/*                                                                         */
/*    void *buf                                                            */
/*       Buffer to read data into                                          */
/*                                                                         */
/*    int count                                                            */
/*       Number of DWORDs to read                                          */
/*                                                                         */
/* Outputs:                                                                */
/*                                                                         */
/*    None                                                                 */
/*                                                                         */
/***************************************************************************/
void insd(word port, void *buf, int count)
{
#ifndef _MSC_VER             /* Not microsoft compiler */

   _ES = FP_SEG(buf);   /* Segment of buf */
```

```
   _DI = FP_OFF(buf);   /* Offset of buf  */
   _CX = count;         /* Number to read */
   _DX = port;          /* Port           */

   // 0x66 below is the operand-size override prefix for x86 processors.
   // Since the 0x66 instruction prefix *complements* the D bit in the
   // segment descriptor, this code assumes that we are running in a 16-bit
   // (DOS?) segment.  It must be modified to work in a 32-bit segment!

   __emit__(0xf3, 0x66, 0x6D); /* asm  REP INSD */
#else
__asm {
       mov            ES,SEGMENT buf;   /* Segment of buf */
    mov        DI,OFFSET buf;    /* Offset of buf  */
    mov        CX,count;         /* Number to read */
       mov            DX,port;          /* Port           */
       REP            INSD;
       }
#endif
}

/***************************************************************************/
/*                                                                         */
/*  OUTSB                                                                  */
/*                                                                         */
/* Purpose: Outputs a string of BYTEs to a hardware port                   */
/*                                                                         */
/* Inputs:                                                                 */
/*                                                                         */
/*    word port                                                            */
/*       hardware port to write to                                         */
/*                                                                         */
/*    void *buf                                                            */
/*       Buffer to write data from                                         */
/*                                                                         */
/*    int count                                                            */
/*       Number of BYTEs to write                                          */
/*                                                                         */
/* Outputs:                                                                */
/*                                                                         */
/*    None                                                                 */
/*                                                                         */
/***************************************************************************/
void outsb(word port, void *buf, int count)
{
#ifndef _MSC_VER             /* Not microsoft compiler */

   _DS = FP_SEG(buf);   /* Segment of buf  */
   _SI = FP_OFF(buf);   /* Offset of buf  */
   _CX = count;         /* Number to read */
   _DX = port;          /* Port           */
   asm   REP OUTSB;
#else
__asm {
       mov            DS,SEGMENT buf;   /* Segment of buf  */
       mov            SI,OFFSET buf;   /* Offset of buf  */
       mov            CX,count;         /* Number to read */
       mov            DX,port;          /* Port           */
       REP            OUTSB;
       }
#endif
}

/***************************************************************************/
/*                                                                         */
/*  OUTSW                                                                  */
/*                                                                         */
/* Purpose: Outputs a string of WORDs to a hardware port                   */
/*                                                                         */
/* Inputs:                                                                 */
/*                                                                         */
/*    word port                                                            */
/*       hardware port to write to                                         */
/*                                                                         */
/*    void *buf                                                            */
/*       Buffer to write data from                                         */
```

```
/*                                                                          */
/*    int count                                                             */
/*       Number of WORDs to write                                           */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*    None                                                                  */
/*                                                                          */
/****************************************************************************/
void outsw(word port, void *buf, int count)
{
#ifndef _MSC_VER             /* Not microsoft compiler */

   _DS = FP_SEG(buf);   /* Segment of buf  */
   _SI = FP_OFF(buf);   /* Offset of buf  */
   _CX = count;         /* Number to read */
   _DX = port;          /* Port           */
   asm   REP OUTSW;
#else
       __asm {
              mov            DS, SEGMENT buf;  /* Segment of buf  */
              mov            SI, OFFSET buf;   /* Offset of buf  */
              mov            CX,count;         /* Number to read */
              mov            DX,port;          /* Port           */
              REP            OUTSW;
       }
#endif
}

/****************************************************************************/
/*                                                                          */
/*  OUTSD                                                                   */
/*                                                                          */
/* Purpose: Outputs a string of DWORDS to a hardware port                   */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    word port                                                             */
/*       hardware port to write to                                          */
/*                                                                          */
/*    void *buf                                                             */
/*       Buffer to write data from                                          */
/*                                                                          */
/*    int count                                                             */
/*       Number of DWORDs to write                                          */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*    None                                                                  */
/*                                                                          */
/****************************************************************************/
void outsd(word port, void *buf, int count)
{
#ifndef _MSC_VER             /* Not microsoft compiler */

   _DS = FP_SEG(buf);   /* Segment of buf  */
   _SI = FP_OFF(buf);   /* Offset of buf  */
   _CX = count;         /* Number to read */
   _DX = port;          /* Port           */

   // 0x66 below is the operand-size override prefix for x86 processors.
       // Since the 0x66 instruction prefix *complements* the D bit in the
       // segment descriptor, this code assumes that we are running in a 16-bit
       // (DOS?) segment.  It must be modified to work in a 32-bit segment!

       __emit__(0xf3, 0x66, 0x6F); /* asm   REP OUTSD; */
#else
       __asm {
              mov            SI, OFFSET buf;   /* Offset of buf  */
              mov            DS, SEGMENT buf;
              mov            cx, count;                   /* num to read */

              mov            DX,port;          /* Port           */
              REP            OUTSD  ;
       }
#endif
```

```
}
```